To appear in *Artificial Intelligence (AIJ)*
http://dx.doi.org/10.1016/j.artint.2016.10.005

Making Friends on the Fly: Cooperating with New Teammates[☆]

Samuel Barrett[a,1,2], Avi Rosenfeld[b], Sarit Kraus[c,d], Peter Stone[e]

[a]*Cogitai, Inc., Anaheim, CA 92808 USA*
[b]*Dept. of Industrial Engineering, Jerusalem College of Technology, Jerusalem, 9116001 Israel*
[c]*Department of Computer Science, Bar-Ilan University, Ramat Gan, 5290002 Israel*
[d]*Institute for Advanced Computer Studies, University of Maryland, College Park MD 20742 USA*
[e]*Dept. of Computer Science, The University of Texas at Austin, Austin, TX 78712 USA*

Abstract

Robots are being deployed in an increasing variety of environments for longer periods of time. As the number of robots grows, they will increasingly need to interact with other robots. Additionally, the number of companies and research laboratories producing these robots is increasing, leading to the situation where these robots may not share a common communication or coordination protocol. While standards for coordination and communication may be created, we expect that robots will need to additionally reason intelligently about their teammates with limited information. This problem motivates the area of *ad hoc teamwork* in which an agent may potentially cooperate with a variety of teammates in order to achieve a shared goal. This article focuses on a limited version of the ad hoc teamwork problem in which an agent knows the environmental dynamics and has had past experiences with other teammates, though these experiences may not be representative of the current teammates. To tackle this problem, this article introduces a new general-purpose algorithm, PLASTIC, that reuses knowledge learned from previous teammates or provided by experts to quickly adapt to new teammates. This algorithm is instantiated in two forms: 1) PLASTIC–Model – which builds models of previous teammates' behaviors and plans behaviors online using these models and 2) PLASTIC–Policy – which learns policies for cooperating with previous teammates and selects among these policies online. We evaluate PLASTIC on two benchmark tasks: the pursuit domain and robot soccer in the RoboCup 2D simulation domain. Recognizing that a key requirement of ad hoc teamwork is adaptability to previously unseen agents, the tests use more than 40 previously unknown teams on the first task and 7 previously unknown teams on the second. While PLASTIC assumes that there is some degree of similarity between the current and past teammates' behaviors, no steps are taken in the experimental setup to make sure this assumption holds. The teammates

[☆]This article contains material from 4 prior conference papers [11–14].

Email addresses: sam@cogitai.com (Samuel Barrett), rosenfa@jct.ac.il (Avi Rosenfeld), sarit@cs.biu.ac.il (Sarit Kraus), pstone@cs.utexas.edu (Peter Stone)

[1]This work was performed while Samuel Barrett was a graduate student at the University of Texas at Austin.

[2]Corresponding author.

were created by a variety of independent developers and were *not* designed to share any similarities. Nonetheless, the results show that PLASTIC was able to identify and exploit similarities between its current and past teammates' behaviors, allowing it to quickly adapt to new teammates.

1. Introduction

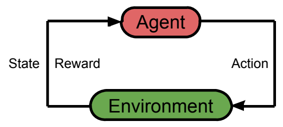
Robots are becoming cheaper and more durable and are therefore being deployed in more environments for longer periods of time. As robots continue to proliferate in this way, many of them will encounter and interact with a variety of other kinds of robots. In many cases, these interacting robots will share a set of common goals, in which case it will be desirable for them to cooperate with each other. In order to effectively perform in new environments and with changing teammates, they should observe their teammates and adapt to achieve their shared goals. For example, after a disaster, it is helpful to use robots to search the site and rescue survivors. However, the robots may come from a variety of sources and may not be designed to cooperate with each other, such as in the response to the 2011 Tohoku earthquake and tsunami [43, 55, 56, 58]. If these robots are not pre-programmed to cooperate, they may not share information about which areas have been searched; or worse, they may unintentionally impede their teammates' efforts to rescue survivors. Therefore, in the future, it is desirable for robots to be designed to observe their teammates and adapt to them, forming a cohesive team that quickly searches the area and rescues the survivors.

This idea epitomizes the spirit of *ad hoc teamwork*. In ad hoc teamwork settings, agents encounter a variety of teammates and try to accomplish a shared goal. In ad hoc teamwork research, researchers focus on designing a single agent or subset of agents that can cooperate with a variety of teammates. The desire is for agents designed for ad hoc teamwork to quickly learn about these teammates and determine how they should act on this new team to achieve their shared goals. Agents that reason about ad hoc teamwork will be robust to changes in teammates in addition to changes in the environment. This article focuses on a limited version of the ad hoc teamwork problem. Specifically, this article investigates how an agent should adapt to new teammates given that it has previously interacted with other teammates and learned from these interactions. However, these past interactions may not be representative of the current teammates.

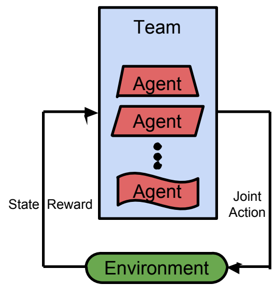
In this article, the word "agent" refers to an entity that repeatedly senses its environment and takes actions that affect this environment, shown visually in Figure 1a. As a shorthand, the terms ad hoc team agent and ad hoc agent are used in this article to refer to an agent that reasons about ad hoc teamwork. The environment includes the dynamics of the world the agent interacts with, as well as defining the observations received by the agent. We treat the other agents in the domain as *teammates* because they share a set of common goals; they are fully cooperative in the terminology of game theory.

Previous work on teamwork has largely assumed that all agents in the domain will act as a unified team and are designed to work with their specific teammates [25, 36, [66, 68]. Methods for coordinating multiagent teams largely rely on specifying standardized protocols for communication as well as shared algorithms for coordination. These approaches do not directly apply to ad hoc teams due to their strong assumptions about this sharing of prior knowledge, which is violated in the ad hoc teamwork scenario. This view of multiagent teams is shown in Figure 1b.

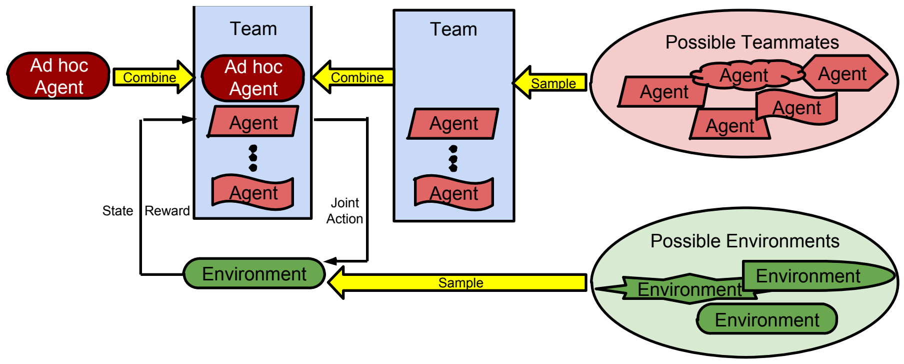
On the other hand, this article will focus on creating a single agent that cooperates with teammates coming from a variety of sources without directly altering the behavior of these teammates. However, all of the agents still share a set of common goals, so it is desirable for them to act as a team. In addition, rather than focusing on a single task, these agents may face a variety of tasks, where a task refers to both the environment other than the team's agents as well as the team's shared goals.



(a) A view of a single agent interacting with its environment used by many reinforcement learning algorithms.



(b) A standard view of a unified team interacting with the environment.



(c) The ad hoc teamwork setting in which an agent cooperates with an ad hoc team of agents to accomplish shared goals on a given environment where the teammates and the environment are each drawn from diverse sets at the beginning of an episode.

Figure 1: Foci of agent based research

The differences of this article from prior work are presented visually in Figure 1. Another existing area of research into how agents should behave is reinforcement learning (RL). Generally, RL problems revolve around a single agent learning by interacting with its environment. In RL problems, agents receive sparse feedback about the quality of sequences of actions. Generally, RL algorithms either model other agents as part of the environment and try to learn the best policy for the single agent given this environment or they consider the case where the whole team is under a single designer's control. In addition, RL algorithms usually learn from scratch in each new environment, ignoring information coming from previous environments. However, there is a growing body of work on applying transfer learning to RL to allow agents to reuse prior experiences on new domains [69]. Figure 1a shows the standard RL view of an agent interacting with its environment. Figure 1b represents a common multiagent view of a unified team interacting with the environment where the agents model their teammates as being separate from the environment. In this case, the team is designed before being deployed to cooperate with these specific agents to interact with a fixed environment. However, these agents rely on knowing their teammates and usually require an explicit communication and/or coordination protocol to be shared among the whole team [36, 53, 73]. On the other hand, this article will focus on ad hoc teams drawn from a set of possible teammates, where the team tackles a variety of possible environments as shown in Figure 1c. In this case, the teammates are not programmed to cooperate with this specific ad hoc agent, and they must be treated as given and inalterable. Instead, this research focuses on enabling the ad hoc agent to cooperate with a variety of teammates in a range of possible environments.

In an ad hoc team, agents need to be able to cooperate with a variety of previously unseen teammates. Rather than developing protocols for coordinating an entire team, ad hoc team research focuses on developing agents that cooperate with teammates in the absence of such explicit protocols. Therefore, we consider a single agent cooperating with teammates that may or may not adapt to its behavior. In this scenario, we can only develop algorithms for the ad hoc team agent, without having any direct control over the other teammates.

In order to be responsive to different teammates and environments, a fully general ad hoc agent needs two general classes of capabilities: 1) the ability to learn how to act in an environment to maximize reward, and 2) the ability to reason about teamwork and learn about its teammates. Previous work in reinforcement learning has largely focused on how an agent should learn about the dynamics of the environment [49, 67], which addresses capability 1. Therefore, this article will leverage this past research and expand this work in the new direction of capability 2: reasoning about the team and social knowledge required for effective teamwork.

Specifically, this article explores a limited version of the full ad hoc teamwork problem in which an ad hoc agent knows the environmental dynamics and encounters unknown teammates, but has previous experience in the domain with other teammates. However, these past experiences may not reflect the current teammates' behaviors. To this end, we introduce a new algorithm for ad hoc teamwork, PLASTIC, that allows an ad hoc agent to reuse knowledge learned about previous teammates to quickly adapt to teammates exhibiting unknown behaviors. We analyze this algorithm in a number of different scenarios in which we vary how similar these previous interactions are to the current teammates' behaviors. The PLASTIC algorithm assumes that there are similarities between the new and old teammates' behaviors, though this may not be true in all scenarios. The experiments in this article *do not* enforce this assumption on the creation of the teammates' behaviors, but PLASTIC still finds similarities to exploit in the scenarios investigated in this article.

This article includes material originally presented in 4 conference papers: [11–14]. In addition to the contributions from those papers, the main new contribution of this article is in showing how the algorithms proposed in these papers form part of an overarching approach, PLASTIC. This article describes the PLASTIC algorithm and how it can be applied to several ad hoc teamwork scenarios where the agent has had previous interactions in the domain. In addition, it expands on the empirical results of these papers as well as introducing a detailed description of the type of problems PLASTIC can solve.

The remainder of this article is organized as follows. Section 2 discusses the problem of ad hoc teamwork as well as the framework and domains used in this article for evaluating ad hoc team agents. We present the background information required to understand the remainder of this article in Section 3. In Section 4, we present PLASTICand its two instantiations (PLASTIC–Model and PLASTIC–Policy). Section 5 describes our empirical analyses of PLASTIC in the pursuit domain, and Section 6 analyzes the performance of PLASTIC for the half field offense in the 2D simulation domain. Section 7 situates this research in literature, and Section 8 contains the concluding remarks and directions for future research.

2. Ad Hoc Teamwork

This section presents the problem of ad hoc teamwork, the problem explored in this article, and the framework used to evaluate ad hoc teamwork in this article. We follow this description by grounding this framework in two domains: the pursuit domain and simulated robot soccer.

2.1. Ad Hoc Teamwork Description

The problem of ad hoc teamwork revolves around how an agent should cooperate with teammates it knows little about. These teammates may not share a communication protocol, but are expected to share the same goals. In the ad hoc team setting, agents can assume that their teammates are attempting to accomplish the same goals, as opposed to game theoretic settings in which agents must reason about ways that opponents can exploit their behaviors. In this article, we assume that the teammates are given; the ad hoc agent does not select its teammates. They may be selected by an earlier decision the agent made or by human intervention. However, the ad hoc agent may not know its teammates' behaviors ahead of time; we investigate how an ad hoc agent can reuse knowledge about past teammates to learn quickly about its new teammates.

This article explores a limited version of the greater ad hoc teamwork problem: how an ad hoc agent should behave when it knows the environmental dynamics and has had prior experiences with other teammates. In general, these past teammates' behaviors may be arbitrarily far from the current teammates' behaviors. However, we hypothesize that there are often similarities in these behaviors that can be exploited to speed up learning.

To better understand the full ad hoc teamwork problem and how this work and past research compares, we identify three dimensions that describe ad hoc teamwork problems. These dimensions affect how the ad hoc agent should behave and the difficulty of the problem it faces, and allow us to more concretely specify which problems this article and past works explore. While there are many possible ways that these scenarios can vary, such as the size of the task's state space and the stochasticity of the domain, we find that the following three dimensions are the most informative for differentiating among the algorithms in existing literature.

1. **Team Knowledge:** Does the ad hoc agent know what its teammates' actions will be for a given state, before interacting with them?
2. **Environment Knowledge:** Does the ad hoc agent know the transition and reward distribution given a joint action and state before interacting with the environment?
3. **Reactivity of teammates:** How much does the ad hoc agent's actions affect those of its teammates?

These dimensions affect the difficulty of planning in the domain in addition to how much an ad hoc agent needs to explore the environment and its teammates. When an ad hoc agent has good knowledge, it can plan without considering exploration, but when it has incomplete knowledge, it must reason about the cost and benefits of exploration. The exploration-exploitation problem has been studied previously, most notably in the reinforcement learning literature, but adding in the need to explore the teammates' behaviors and the ability to affect them considerably alters this tradeoff. We believe that scenarios with lower amounts of team knowledge and environment knowledge and higher amounts of teammate reactivity are more representative of the full, ad hoc teamwork problem. We expect that agents that are capable of dealing with these concerns are robust, ad hoc team agents and are able to deal with nearly any ad hoc teamwork scenario. Sections 2.1.1–2.1.3 provide further details about each of these dimensions, how we measure them, and why they are important for ad hoc teamwork. Note that while all of the equations in the following sections use summations, these summations can be converted to integrals for domains with continuous states or actions.

We use these dimensions in this article to characterize our experiments (Sections 2.4.3 and 2.5.2) as well as related work (Section 7.5.1). We hypothesize that problems with similar characteristics along these dimensions can be approached with similar algorithms. The PLASTIC algorithm introduced in this article addresses a subset of such problems. In Section 2.2, we specify that subset.

2.1.1. Team Knowledge

The ad hoc agent's knowledge about its teammates' behaviors gives insight into the difficulty of planning in the domain. The agent's knowledge can range from knowing the complete behaviors of its teammates to knowing nothing about them. Settings with partial information are especially relevant, because in many real world problems, the exact behavior of a teammate may not be known, but some reasonable guidelines of their behaviors exist. For example, when playing soccer, one can usually assume that a teammate will not intentionally pass to the other team or shoot at the wrong goal. If the behaviors are completely known, the agent can reason fully about the team's actions, while if the behaviors are unknown, the agent must learn about them and adapt to find a good behavior.

To estimate the ad hoc agent's knowledge about its teammates' behaviors, we compare the actions the ad hoc agent expects them to take and the ground truth of what actions they take. Specifically, we compare the expected distribution of teammate actions to the true distribution that the teammates follow. To compute the difference between the distributions, we use the Jensen-Shannon divergence measure, which was chosen because it is a smoothed, symmetric variant of the popular Kullback-Leibler divergence measure. When the ad hoc agent has no information about a teammate's action, we assume that it uses the uniform distribution to represent its actions. Therefore, we define the knowledge measure as

$$K(T, \text{Pred}) = \begin{cases} 1 & \text{if JS}(T, \text{Pred}) = 0 \\ 1 - \dfrac{\text{JS}(T, \text{Pred})}{\text{JS}(T, U)} & \text{if JS}(T, \text{Pred}) < \text{JS}(T, U) \\ -\dfrac{\text{JS}(\text{Pred}, U)}{\text{JS}(U, \text{Point})} & \text{otherwise} \end{cases} \quad (1)$$

where T is the true distribution, Pred is the predicted distribution, U is the uniform distribution, Point is a distribution with all weight on one point (e.g. $[1, 0, 0, \ldots]$), and JS is the Jensen-Shannon divergence measure. By this definition, $K(T, T) = 1$, so the knowledge is complete if the ad hoc agent knows the true distribution. $K(T, U) = 0$, representing when the ad hoc agent has no knowledge and relies on the uniform distribution. Finally, if the predicted distribution is less accurate than the uniform distribution, then $K(T, \text{Pred})$ is negative, with a minimum value of -1. This measure captures the range [0,1] smoothly, but can still be used for the range [-1,0] [3]. However, we generally expect the prediction to be a higher entropy distribution than the true distribution as the ad hoc agent ought to correctly model its uncertainty in its teammates' behaviors rather than being confident and wrong, which keeps the measure in the range [0,1].

We define the ad hoc agent's knowledge about its teammates' behaviors as

$$\text{TeamK} = \frac{1}{nk} \sum_{s=1}^{n} \sum_{t=1}^{k} K(\text{TrueAction}_t(s), \text{PredAction}_t(s))$$

where $1 \leq s \leq n$ is the state, $1 \leq t \leq k$ specifies a teammate, $\text{TrueAction}_t(s)$ is the ground truth action distribution for teammate t for state s, and $\text{PredAction}_t(s)$ is the action distribution that the ad hoc agent predicts that teammate t will select for state s.

We assume that $\text{PredAction}_t(s)$ is the uniform distribution if the agent has no information about teammate t's actions in state s. Thus, if the ad hoc agent has better information about its teammates' behaviors, the distance between the distributions will be smaller and TeamK will be higher. If the ad hoc agent can limit possible actions that its teammates may take or bias its predictions towards more likely actions, TeamK will be higher. For instance, if there are 100 possible actions and the ad hoc agent can narrow a teammate's action to two choices, TeamK will be relatively large.

2.1.2. Environmental Knowledge

Another informative dimension is how much knowledge the ad hoc agent has about the effects of a joint action given a state, for example the transition and reward func-

[3] One slight anomaly of this measure is that when T is the uniform distribution (e.g. [.5,.5]), K is either 1 when Pred is exactly correct at [.5 .5] or negative. For all other values of T, K smoothly spans the range [-1,1].

tions. If the ad hoc agent has complete knowledge about the environment, it can plan about what actions it should select more simply than if it must also consider unknown effects of actions. However, if it has incomplete knowledge, it must explore its actions and face the standard problem of balancing exploring the environment versus exploiting its current knowledge.

Similarly to teammate knowledge, we formally define the ad hoc agent's knowledge about the environment's transitions as

$$\text{TransK} = \frac{1}{nm} \sum_{s=1}^{n} \sum_{j=1}^{m} K(\text{TrueTrans}(s,j), \text{PredTrans}(s,j))$$

where $1 \leq s \leq n$ is the state, $1 \leq j \leq m$ is a joint action, K is taken from Equation (1), $\text{TrueTrans}(s,j)$ is the ground truth transition distribution from state s given joint action j, and PredTrans is the ad hoc agent's predicted transition distribution. If the agent has no information about the transitions, we assume that $\text{PredTrans}(s,j)$ is the uniform distribution. Intuitively, if the ad hoc agent knows more about the transition function, then the distance between TrueTrans and PredTrans will be smaller and as a result TransK will be higher. We define the agent's knowledge about the environmental rewards similarly, and let $\text{EnvK} = (\text{TransK}, \text{RewardK})$.

2.1.3. Teammate Reactivity

The optimal behavior for the ad hoc agent also depends on how much its teammates react to its actions. If its teammates' actions do not depend on the ad hoc agent at all, the ad hoc agent can simply choose its actions to maximize the team reward, as if it were a single agent problem. Considering the actions of its teammates separately from that of the environment may still help computation by factoring the domain. However, if the teammates' actions depend strongly on the ad hoc agent's actions, the ad hoc agent's reasoning should consider what its teammates' reactions will be. If the ad hoc agent is modeling its teammates and its teammates are modeling the ad hoc agent, the problem can become recursive, as is directly addressed by Gmytrasiewicz et al.'s Recursive Modeling Method [35].

A formal measure of the teammate reactivity needs to capture how different the teammates' actions will be when the ad hoc agent chooses different actions. We measure the distance between the resulting distributions of the teammate joint actions, using the pairwise Jensen-Shannon divergence measures. However, it is desirable for the distance to be 1 when the distributions have no overlap, so we use a normalizing constant of $\log 2$. Thus, we define the *reactivity* of a domain in state s as

$$\text{Reactivity}(s) = \frac{1}{(m-1)^2 \log 2} \sum_{a=1}^{m} \sum_{a'=1}^{m} \text{JS}(T(s,a), T(s,a'))$$

where JS is the Jensen-Shannon divergence measure, $1 \leq a, a' \leq m$ is the actions available to the ad hoc agent, and $T(s,a)$ is the distribution of the teammates' joint actions given the state s and the ad hoc agent's action, a. We use $m-1$ in the denominator because we exclude the case where $a = a'$; in the numerator, the JS measure will be 0 in this case. For the overall reactivity of the domain, we average over the states, resulting in $\text{Reactivity} = \frac{1}{n}\sum_{s=1}^{n} \text{Reactivity}(s)$. It is possible to consider how an action affects the teammates' actions further in the future, but we restrict our focus to one step reactivity for this paper.

Note that all of the sums in this formulation can be converted to integrals for continuous states or actions. This formulation is similar to the empowerment measure used by Jung et al. [47], but we consider the ad hoc agent's ability to change the actions of its teammates rather than the environment state.

2.2. Problem Description

In this article, we focus on exploring the dimension of teammate knowledge. Specifically, we explore settings in which the ad hoc agent has prior experiences with past teammates and is trying to use knowledge from these experiences to quickly adapt to new teammates. We assume that the teammates B are drawn from a set of possible teammate types A. Then, the question is how an ad hoc agent a should cooperate with these teammates. However, the ad hoc agent may not know A or B. We explore different amounts of prior knowledge given to the ad hoc agent. We primarily focus on the setting in which the ad hoc agent has previously observed teams P operating in the current domain. While it does not have access to their full decision-making process, the ad hoc agent can learn about their behaviors by observing the world states and the teams' actions. We hypothesize that the ad hoc agent can use the knowledge it learns from these interactions to quickly learn to adapt to new teammates' behaviors by exploiting similarities between these behaviors.

In this article, we assume that all of the teammates and the ad hoc agent share a common goal. In addition, we do not explore the environmental knowledge dimension; we assume that the ad hoc agent fully knows the domain. Furthermore, we assume that there is no shared protocol for explicit communication. Specifically, the ad hoc agent cannot directly communicate its intentions to its teammates, and the agents can only communicate through their movements in the world. Finally, we assume that the teammates' behaviors from A and P are stationary; the teammates only respond to the ad hoc agent's immediate actions and do not change their behaviors and learn over time. As such, the algorithms put forth in this article address only a small subset of possible ad hoc teamwork scenarios leaving much room for fruitful future research.

The difficulty of the problem we do consider is that the ad hoc agent does not have full knowledge of its teammates' behaviors, though it does have prior experiences with other teammates. Nor does the agent have a shared communication protocol that allows those teammates to explicitly communicate their intentions. Therefore, the ad hoc agent must observe its teammates to determine their behaviors. Once it knows the behaviors its teammates exhibit, the ad hoc agent can adapt accordingly. To speed up the process of determining the teammates' behaviors, the ad hoc agent can draw upon its observations of past teammates, exploiting similarities between the current and past teammates' behaviors.

In this article, we consider scenarios in which the ad hoc agent has different amounts of prior knowledge of its teammates. In terms of the dimensions of ad hoc teamwork, we are exploring the team knowledge dimension, varying the accuracy of PredAction and thus the value of TeamK. We consider the scenario where $P = A$, so the ad hoc agent has seen all of the potential teammate types, but does not know which team was drawn from A. We also consider the more complex scenario, where P and A do not overlap, i.e. $P \cap A = \emptyset$. This scenario occurs when the ad hoc agent has previously attempted the task with other types of teammates, but not with the current teammate type. When $P \cap A = \emptyset$, the accuracy of PredAction is lower, and thus the ad hoc agent's TeamK is lower.

In all of the scenarios in this article, TeamK is still relatively high given the ad hoc agent's previous experiences and the fact that the current teammate types are similar to the past teammates. Note that while this similarity is assumed, it is *not* enforced by the authors. Specifically, we use agent behaviors from a variety of independent developers, and the results of PLASTIC show that there are similarities to exploit in the scenarios explored in this article. We hypothesize that these sort of natural similarities occur in many settings. While considering an ad hoc agent with no knowledge of its teammates is interesting, we think that having some prior knowledge in the form of experience of past teammates exhibiting other behaviors is more realistic. Agents will encounter a number of teammates over their lifetime, and they should be able to draw upon these experiences to learn more quickly when encountering new teammate types that have similarities to these past teammate types.

In this article, we focus on the scenario where the ad hoc agent knows its environment, i.e. that EnvK $= (1, 1)$. We believe that expanding this work to scenarios where the ad hoc agent must simultaneously learn about its environment is an important avenue for future research. In terms of the dimensions, we believe that future research should focus on how ad hoc agents should behave in scenarios that have low values of both TeamK and EnvK.

In addition, this article focuses on problems with limited reactivity of its teammates. While the agents do react to the ad hoc agent's actions, they do not learn from it over time. The ad hoc agent's actions have limited effects on its teammates; hence, the values of Reactivity are low to moderate for the domains in the article. We believe that future research into teammates that learn about the ad hoc agent is needed; ad hoc agents should be able to deal with higher amounts of teammate reactivity.

2.3. Evaluation Framework

Directly measuring teamwork is far from straightforward. In many cases, the only easily measurable aspect is the overall performance of the team, which makes it difficult to assign credit to each agent. By placing an agent on a variety of teams and measuring those teams' performances, we can estimate how good the agent is at teamwork.

Therefore, we adopt the evaluation framework introduced by Stone et al. [63] which evaluates an ad hoc team agent while considering the teammates and domains it may encounter. This framework is specified in Algorithm 1. According to this framework, the performance of the ad hoc team agent a depends on the distribution of problem domains D and the distribution of possible teammates A that it will cooperate with. For the team B cooperating to execute the task d, $s(B, d)$ is a scalar score representing their effectiveness, where higher scores indicate better performance. The algorithm takes a sampling approach to average the agent's performance across a range of possible tasks and teammates to capture the idea that a good ad hoc team player ought to be robust to a wide variety of teamwork scenarios. We use s_{min} as a minimum acceptable reward for the team to be evaluated, because the ad hoc team agent may be unable to accomplish a task if its teammates are too ineffective, regardless of its own abilities. It is mainly used to reduce the number of samples required to evaluate the ad hoc agents and reduces the noise in the comparisons. Metrics other than the sum of the rewards can be used depending on the domain, such as the worst-case performance.

Algorithm 1 Ad hoc agent evaluation

```
1:  function Evaluate:
        inputs:
            $a$                         ▷ the ad hoc agent
            $A$                         ▷ the set of possible teammate agents
            $D$                         ▷ the set of possible domains
        outputs:
            $\frac{r}{n}$               ▷ the average performance (reward)
        params:
            $s_{min}$                   ▷ the minimal acceptable performance of a team
            $n$                         ▷ the number of iterations

2:      Initialize: $r = 0$
3:      for $i = 1$ to $n$ do
4:          Sample a task $d$ from $D$
5:          Randomly draw a subset of agents $B$, from $A$ such that $E[s(B, d)] \geq s_{min}$
6:          Randomly select one agent $b \in B$
7:          Create the new team $C = \{a\} \cup B \backslash \{b\}$
8:          $r = r + s(C, d)$
9:      return $\frac{r}{n}$

10: If Evaluate$(a_0, A, D) >$ Evaluate$(a_1, A, D)$ and the difference is significant, we
    can conclude that $a_0$ is a better ad hoc team agent than $a_1$ in domains $D$ over the
    set of possible teammates $A$.
```

2.4. Pursuit Domain

As discussed in the previous section, the evaluation of ad hoc team agents depends strongly on the domains that they may encounter. Therefore, this section describes the first domain that is used for evaluating ad hoc team agents in this article. The pursuit domain, also known as the predator-prey domain, is a popular problem in multiagent systems literature as it requires cooperation between all of the teammates to capture the prey while remaining simple enough to evaluate approaches well [66]. There are many versions of the pursuit domain with different rules, 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 article, 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 2, and videos of the domain can be viewed online.[4]

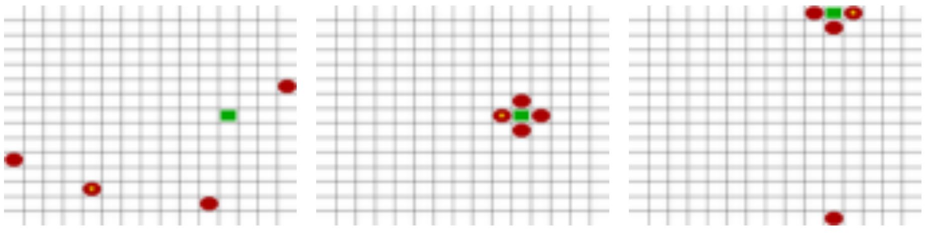

(a) Random starting position. (b) A valid capture position. (c) A second valid capture position.

Figure 2: 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.

2.4.1. Hand-Coded Teammates

In order to meaningfully test the proposed ad hoc teamwork algorithms, four hand-coded predator algorithms with varying and representative properties were used. The *greedy* (GR) predator moves towards the nearest open cell that neighbors the prey, ignoring its teammates' actions. On the other hand, the *teammate-aware* (TA) predator considers its teammates, and allows the predator that is farthest from the prey have the cell closest to it. In addition, the teammate-aware predator uses the A* path planning algorithm to select its actions while the greedy predator only considers immediate collisions. It is expected that the differences between these teammates will require the ad hoc agent to adapt and reason about how its actions will influence its teammates' actions. In addition to these two deterministic agents, two stochastic agents are used that each select an action distribution at each time step. The *greedy probabilistic* (GP) predator moves similarly to the greedy predator except that it has a chance of taking a longer path to the greedy destination. Finally, the *probabilistic destinations* (PD) predator chooses a new destination near the prey at every time step, slowly encircling the prey before converging on it.

These behaviors were chosen to provide a spread of representative behaviors. The deterministic GR predator largely ignores its teammates' actions while the deterministic TA 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. It is expected that the ad hoc agent will need to cooperate differently with these two types of agents based on their reactivity. In addition to these two deterministic agents, we use the stochastic GP and PD agents. We expect it to be fairly trivial for the ad hoc agent to differentiate the deterministic agents, but harder to differentiate the stochastic agents. Therefore, the ad hoc agent will be forced to reason about the uncertainty of its teammates' behaviors for longer. Furthermore, these behaviors are significantly different from the deterministic behaviors,

[4]http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_Pursuit

and interacting with them requires reasoning about noise in future outcomes.

2.4.2. Externally-created Teammates

While the set of hand-coded teammates attempts to be representative, this set is limited and possibly biased as the agents were designed by someone thinking about ad hoc teamwork. Therefore, we also consider externally-created teammates to provide a broader range of agents created by developers not planning for ad hoc teamwork scenarios. Specifically, we use two additional sets of teammates in this article, both created by undergraduate and graduate computer science students. These agents were created for an assignment in two workshops on agent design with no discussion of ad hoc teams; instead, 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. Both sets of agents come from a workshop taught by Sarit Kraus at Bar Ilan University, one taught in the spring of 2010, and the other taught in the spring of 2011. The first set of agents contains the best 12 student agents taken from the first class of 41 students, filtered by their ability to capture a randomly moving prey in a 5x5 world in less than 15 steps on average (i.e. $s_{min} = 15$ in Algorithm 1). This set of agents is called Student$_{\text{Selected}}$. The second set of agents, Student$_{\text{Broad}}$, comes from a second offering of the course and contains 29 agents from a class of 31 students. Student$_{\text{Broad}}$ contains a wider range of performance than Student$_{\text{Selected}}$ as it is filtered less heavily. The better quality of agents in Student$_{\text{Broad}}$ is due to the improvements to the directions and architecture provided to the second class of students based on the lessons learned from the first offering of the course. The only filtering of this set was removing one student team for never capturing the prey and a second for taking excessively long computation time.

2.4.3. Dimension Analysis

Given the problem description, we can analyze how the pursuit domain is described by the dimensions introduced in Section 2.1. The ad hoc agent's knowledge about its team (TeamK) varies in the different tests, as does the reactivity of the teammates. When the ad hoc agent knows its teammates' behaviors, TeamK $= 1$. A variety of other scenarios are summarized in Table 1, where we vary the type of teammates as well as the prior knowledge the ad hoc agent has about its teammates. The ad hoc agent completely knows the environment dynamics, leading to EnvK $= (1, 1)$.

Teammate Type	Prior Knowledge	World Size	TeamK	Reactivity
Hand-coded	Hand-coded	5x5	0.719	0.717
Hand-coded	Hand-coded	20x20	0.360	0.801
Student$_{\text{Selected}}$	Hand-coded	20x20	0.156	0.801
Student$_{\text{Selected}}$	Known Learned Set	20x20	0.318	0.801
Student$_{\text{Broad}}$	Hand-coded	20x20	0.098	0.800
Student$_{\text{Broad}}$	Known Learned Set	20x20	0.301	0.800
Student$_{\text{Broad}}$	Leave-one-out Known Learned Set	20x20	0.280	0.800

Table 1: TeamK and Reactivity for various settings in the pursuit domain.

The moderate reactivity values for most teammate types implies that it is vital to understand and model the ad hoc agent's teammates. However, the teammates do not learn over time from the ad hoc agent's actions. The lower values of team knowledge for the student teammates shows the importance of quickly narrowing the field of models to descriptive ones to allow for better planning. This hypothesis and approaches to learning about teammates are explored empirically in Section 5.

2.5. *Half Field Offense in the 2D RoboCup Simulator*

While the pursuit domain provides an interesting multiagent domain for testing teamwork, it is still fairly simple compared to real world problems. In order to test the scalability of our approach, it is important to also consider more complex problems. Therefore, we also consider a simulated robot soccer domain used in the 2D RoboCup Simulation League.

The 2D Simulation League is one of the oldest leagues in RoboCup and is therefore one of the best studied, both in competition and research. In this domain, teams of 11 autonomous agents play soccer on a simulated 2D field and must choose their actions every 100 ms. Before each action, the agents receive noisy sensory information such as their location, the location of the ball, and the locations of nearby agents. After processing this information, agents select abstract actions that describe how they move in the world, such as dashing, kicking, and turning. The 2D simulation server and the full manual that includes the perception and action models can be found online.[5] This domain is used as it provides a testbed for teamwork in a complex domain without requiring focus on areas such as computer vision and legged locomotion.

Rather than use full 10 minute 11 on 11 game, this article instead uses the quicker task of half field offense introduced by Kalyanakrishnan et al. [48]. In Half Field Offense (HFO), a set of offensive agents attempt to score on a set of defensive agents, including a goalie, without letting the defense capture the ball. A view of this game is shown in Figure 3, and more information and videos can be found online.[6] This task is useful as it allows for much faster evaluation of team performance than running full games as well as providing a simpler domain in which to focus on ways to improve ball control. In this article, we consider two versions of the HFO domain: 1) a *limited version* with two offensive agents and two defensive agents including the goalie and 2) the *full version* with four offensive agents and five defensive agents including the goalie. Videos of both versions of this domain can be viewed online.[7] Hausknecht et al. [39] has built upon the work in this article and released an open source version of the half field offense domain.[8]

If the ball leaves the offensive half of the field or the defense captures the ball, the offensive team loses. If the offensive team scores a goal, they win. In addition, if no goal is scored within 500 simulation steps (50 seconds), the defense wins.

At the beginning of each episode, the ball is moved to a random location within the 25% of the offensive half closest to the midline. Let *length* be the length of the

[5] http://sourceforge.net/projects/sserver/
[6] http://www.cs.utexas.edu/~AustinVilla/sim/halffieldoffense/
[7] http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO
[8] https://github.com/LARG/HFO

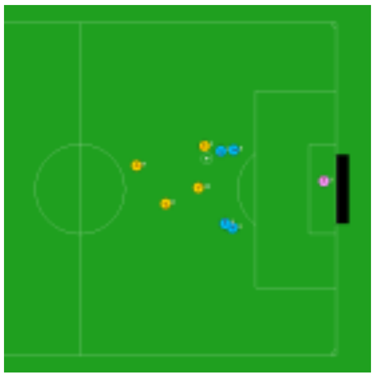


Figure 3: A screenshot of half field offense in the 2D soccer simulation league. The yellow agent number 11 is under our control, and remaining yellow players are its externally created teammates. These agents are trying to score against the blue defenders.

soccer pitch. Offensive players start on randomly selected vertices forming a square around the ball with edge length $0.2 \cdot \textit{length}$ with an added offset uniformly randomly selected in $[0, 0.1 \cdot \textit{length}]$. The goalie begins in the center of the goal, and the remaining defensive players start randomly in the back half of their defensive half.

2.5.1. Externally-created Teammates

Unlike in the pursuit domain, we do not use hand-coded teammates as it is difficult to define what a set of representative policies might be in this domain. Instead, it is more productive to consider agents created for the RoboCup competition. It is expected that these agents represent a far better spread of possible behaviors than any hand-coded teammates, given the years of improvements implemented for the competitions.

As part of the 2D simulation league competition, teams are required to release binary versions of their agents following the competition. Therefore, we use the binary releases from the 2013 competition held in Eindhoven, Netherlands.[9] These agents provide an excellent source of *externally-created* teammates with which to test the possible ad hoc team agents. Specifically, we use 6 of the top 8 teams from the 2013 competition, omitting 2 as they do not support playing games faster than real time. In addition, we use the team provided in the code release by Helios [3], commonly called agent2d, which serves as the basis of many teams in the league. Therefore, there are a total of 7 possible teams that our agent may encounter: agent2d, aut, axiom, cyrus, gliders, helios, and yushan.

In order to run some teams used in the RoboCup competition, it is necessary to field the entire 11 player team for the agents to behave correctly. Therefore, it is necessary to create the entire team and then constrain the additional players to stay away from play, only using the agents needed for half field offense. These additional players are moved to the other side of the field every time step. This approach may affect the players used in the HFO, but our initial tests showed that the teams still perform well. We choose a

[9] `http://www.socsim.robocup.org/files/2D/`

fixed set of player numbers for the teammates, based on which player numbers tended to play offensive positions in our observations of their play in initial experiments. In the limited HFO task, defensive players use the helios behavior, while in the full HFO task, they use the agent2d behavior.

2.5.2. Dimension Analysis

In order to better understand the properties of the half field offense domain and the teammates that the ad hoc agent may encounter, we can use the dimensions described in Section 2.1. We approximate the Jensen-Shannon divergence measure using Monte Carlo sampling. Recall from Section 2.1 that $\text{JS}(P, Q) = \frac{1}{2}(\text{KL}(P, M) + \text{KL}(Q, M))$ where $M = \frac{1}{2}(P + Q)$ and the Kullback-Leibler divergence is defined as

$$\text{KL}(P, M) = \int P(X) \log \frac{P(x)}{M(x)}$$

The Monte Carlo approximation is given by

$$\widehat{\text{KL}}(P, M) = \frac{1}{n} \sum_{i}^{n} \log \frac{P(x_i)}{M(x_i)}$$

where $M(x_i) = \frac{1}{2}(P(x_i) + Q(x_i))$. As the number of samples goes to infinity, this approximation converges to the true value of KL.

Given that the actions are also continuous, we need to consider an infinite number of joint actions. In addition, the ad hoc agent does not directly observe the actions of its teammates. Therefore, we use the resulting locations of the agents as an estimate of their actions. We assume that the effects of these actions are noisy modeled with a Gaussian distribution with standard deviation of 40 and 40° for distances and angles respectively.

Applying this methodology with the calculation introduced in Section 2.1.1 leads us to an approximate value of 0.425 for TeamK in the limited HFO task. In this task, the ad hoc agent knows that its teammate's behavior is drawn from the set of 7 potential behaviors. In the full HFO task, this methodology calculates that TeamK ≈ 0.295.

We can similarly approximate the value of Reactivity by using the calculation introduced in Section 2.1.3. Given that the 2D RoboCup simulator is open source and all domain parameters are passed on to the players, the ad hoc agent completely knows the environment dynamics. In addition, note that the opponents' behaviors are known by the ad hoc agent. Therefore, EnvK $= (1, 1)$. However, it is worth noting that it is complex to model the full domain, so in our tests, the ad hoc agent does not explicitly model the HFO dynamics. The 7 possible teams that the ad hoc agent may encounter have an average reactivity of Reactivity $= 0.263$ in the limited HFO task and Reactivity $= 0.507$ in the full version of the task.

The moderate reactivity means that the ad hoc agent can help its team and should consider how its actions affect its teammates, especially in the full HFO domain. However, it also means that the teammates have limited ability to change based on the ad hoc agent's actions. For instance, they are not learning from the ad hoc agent over time. In addition, the perfect environmental knowledge means that the agent does not need to explore the environment. On the other hand, the lower teammate knowledge means that it is helpful to explore the teammates' behaviors, especially in the full HFO domain, where the space of its teammates' behaviors is larger. Notice that these values are close to those arising from the pursuit domain. Therefore, we once again expect that a similar approach should be effective in this domain. However, the complexity of fully modeling the domain means that methods applied to the other two domains may run into issues here. Therefore, we expect that a model-free approach may be more effective, but using teammate knowledge similarly should be effective. The model-free approach is analyzed in more depth with empirical results in Section 6.

3. Background

While the previous section describes the general problem investigated in this article, this section describes the mathematical model used to analyze this problem. In addition, this section presents the existing algorithms that our approach builds upon.

3.1. Markov Decision Process

Agents that need to cooperate in ad hoc teams need to handle sequential decision making problems; therefore, we choose to model these problems as Markov Decision Processes [67]. An MDP is 4-tuple (S, A, P, R), where S is a set of states, A is a set of actions, $P(s'|s, a)$ is the probability of transitioning from state s to s' when after taking action a, and $R(s, a)$ is a scalar reward given to the agent for taking action a in state s. In both domains, $s \in S$ corresponds to the current positions of every agent and $a \in A$ is the action that the ad hoc agent chooses. In this framework, a policy π is a mapping from states to actions, which defines an agent's behavior for every state. The agent's goal is find the policy that maximized its long term expected rewards. For every state-action pair, $Q^*(s, a)$ represents the maximum long term reward that can be obtained from (s, a) and is defined by solving the Bellman equation

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$$

where $0 < \gamma < 1$ is the discount factor representing how much more immediate rewards are worth compared to delayed rewards. The optimal policy π^* can then derived by choosing the action a that maximizes $Q^*(s, a)$ for every $s \in S$.

Once we model a problem as an MDP, it becomes clear what the agent's objective is: to maximize long term expected reward. In our setting this translates into the ad hoc agent optimally cooperating with its teammates to accomplish their shared goals. There are a number of ways to calculate the actions that result in the best long term expected reward. Value Iteration (VI) [67] is a dynamic programming approach to solve this problem exactly. Monte Carlo Tree Search (MCTS) algorithms such as Upper Confidence Bounds for Tree (UCT) [51] approximately calculate the actions that maximize rewards using sampling. While VI requires a full model of the teammates' actions and the domain's transition function, MCTS algorithms only require a way of sampling the results of the agent's actions. In addition, MCTS algorithms are often more computationally tractable on large domains than VI. A popular approach that does not require any explicit modeling of the domain is the Fitted Q Iteration (FQI) algorithm introduced by Ernst et al. [28]. Similar to VI, FQI iteratively backs up estimates of the values of state-action pairs. However, FQI employs samples of states and the outcomes of actions to approximate these values, which allows it to handle continuous domains.

These algorithms are explained in greater depth in Appendix A. Other algorithms for solving MDPs can be used in PLASTIC, but the results in Sections 5 and 6 employ these methods.

3.2. Transfer Learning

While the previous section discusses methods for how an ad hoc agent can compute a policy for cooperating with its teammates given a model of its teammates, it does not specify where these models come from. An approach that we employ in this article is to learn models of past teammates, treating it as a supervised learning problem. When the ad hoc agent has a limited amount of experiences with its current teammates in addition to extensive experiences with past experiences, it may be able to learn models specific to the current teammates. Unfortunately, the limited experiences with the current teammates makes learning a new model from scratch infeasible. However, it may be able to reuse information it has learned about past teammates in addition to what it knows of its current teammates to learn a new model of its teammates, an idea synonymous with transfer learning.

In Transfer Learning (TL), the goal is to reuse information learned on a *source* data set to improve results on a *target* data set. For TL, only the performance on the target data matters; the source data is only used for training. Following this terminology, for ad hoc teamwork settings, we consider the current teammates to be the *target* set, and the previously observed teammates are the *source* set.

The transfer learning algorithm, TwoStageTransfer [14], used in this article is capable of transferring knowledge from multiple sources. TwoStageTransfer is inspired by the TwoStageTrAdaBoost algorithm [57], and it is designed to explicitly leverage multiple source data sets. TwoStageTransfer's goal is to find the best possible weighting for *each* set of source data and create a classifier using these weights. Rather than trying all of possible weightings, TwoStageTransfer first evaluates each data source independently, and calculates the ideal weight of that data source using cross validation. Then, it greedily 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 when combined 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.

4. Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC)

This section introduces the Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC) algorithms that enable an ad hoc team agent to cooperate with a variety of different teammates. 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, if the teammates' behaviors are suboptimal, this approach will limit how much the ad hoc agent can help its team. Therefore, in this article, we adopt the approach of modeling possible teammate behaviors and planning how to act based on these models. This approach allows an ad hoc agent to reason about how well its models predict its teammates' actions and then allows it to convert these predictions into the actions it needs to take to accomplish its goals. If the models are correct and the ad hoc agent is given enough time to plan, this approach will lead to optimal performance of the ad hoc agent, helping its team achieve the best outcome, as algorithms such as UCT provably converge to the optimal behavior. Note that this may not be the optimal performance of any team, but it is optimal for the ad hoc agent given that the behaviors of its teammates are fixed.

4.1. Overview

A visual overview of PLASTIC is given in Figure 4. The short summary of the approach is that the ad hoc agent either learns about a set of prior teammates or is given some hand-coded information about possible teammates. Then, the agent uses this prior knowledge to select its actions and update its beliefs about its teammates by observing their reactions to its behavior.

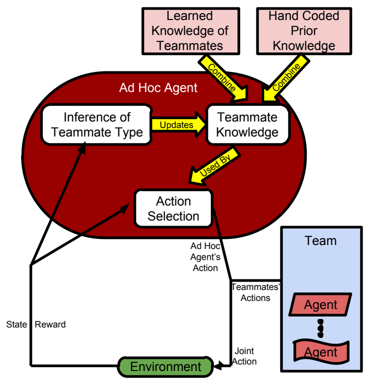


Figure 4: Overview of using PLASTIC to cooperate with unknown teammates.

In this article, this general approach is realized in two algorithms. One algorithm, PLASTIC–Model, focuses on a model-based approach. In this approach, the ad hoc agent learns models of its past teammates, selects which models best predict its current teammates, and then uses these models to plan how to act in order to cooperate with the teammates. The second algorithm is called PLASTIC–Policy and uses a model-free approach. In this variant, the ad hoc agent learns a policy to cooperate with each of its past teammates, selects which policies best match how to cooperate with its current teammates, and then selects actions using these policies. These two algorithms are described in the remainder of the section. This general approach is specified in Algorithm 2. The subroutines LearnAboutPriorTeammate, SelectAction, and UpdateBeliefs are described for each of the two algorithms in the following section.

As shown in Algorithm 2, PLASTIC begins by initializing its knowledge using the provided prior knowledge and what it has learned about previous teammates in Lines 2–5.

Algorithm 2 Pseudocode of PLASTIC

```
1: function PLASTIC:
       inputs:
           PriorTeammates          ▷ past teammates the agent has encountered
           HandCodedKnowledge      ▷ prior knowledge coded by hand
           BehaviorPrior           ▷ prior distribution over the prior knowledge
   ▷ initialize knowledge using information from prior teammates
2:     PriorKnowledge = HandCodedKnowledge
3:     for $t \in$ PriorTeammates do
4:         PriorKnowledge = PriorKnowledge $\cup$ {LearnAboutPriorTeammate($t$)}
5:     BehaviorDistr = BehaviorPrior(PriorKnowledge)      ▷ initialize beliefs

   ▷ act in the domain
6:     Initialize $s$
7:     while $s$ is not terminal do
8:         $a$ = SelectAction(BehaviorDistr, $s$)
9:         Take action $a$ and observe $r, s'$
10:        BehaviorDistr = UpdateBeliefs(BehaviorDistr, $s, a$)
```

LearnAboutPriorTeammate is defined differently for the two variants, but in both algorithms it learns information about the prior teammate, encoding the knowledge to be used in the SelectAction subroutine. Lines 6–10 show how PLASTIC selects the agent's actions. PLASTIC updates its beliefs over the teammate models or policies by observing their actions and using the UpdateBeliefs function implemented in the two variants.

The core difference between PLASTIC–Model and PLASTIC–Policy is that the former uses a model based approach, while the latter uses a policy based approach. The pursuit domain has $400^5 \approx 10^{13}$ states and has nearly deterministic actions, as the only non-determinism is in reconciling collisions. The half field offense domain has continuous states leading to an effectively infinite base and more agents leading to a higher exponent. In addition to this higher-dimension state space, the actions are non-deterministic, leading to a much higher branching factor. The actions are also continuous, though we have discretized the ad hoc agent's actions. Combining this larger, continuous state space and the higher branching factor hampers the ability to search for the best policy for the ad hoc agent. However, learning policies for behaving in this type of complex domains is possible. Specifically, approaches such as Fitted Q Iteration can handle learning policies in large, continuous state spaces. Therefore, PLASTIC–Policy is capable of tackling more complex problems than PLASTIC–Model.

On the other hand, PLASTIC–Model is more scalable to diverse teams. It stores a model of each type of teammate behavior, rather than each team. On the other hand, PLASTIC–Policy stores a policy for cooperating with a team, which depends on the behavior of each teammate. In addition, PLASTIC–Model is better able to handle situations (not explored in this article) in which its models of teammate behaviors change over time, such as when ongoing learning changes these models. As these models change, PLASTIC–Model can simply adapt by planning a new behavior using these models, while PLASTIC–Policy requires recalculating the policy for the resulting MDP. On the other hand, the main advantage of PLASTIC–Policy is its ability to scale to more complex problems.

4.2. *PLASTIC–Model*

When an agent has a good model of its *environment*, it can use this model to plan good actions using a limited number of interactions with the environment. For an ad hoc agent to plan, it also needs to model its *teammates*; therefore, it is useful for the ad hoc agent to build models of its teammates' behaviors. Given that learning new models online takes many samples, it is useful to reuse information learned from past teammates. This section describes PLASTIC–Model, a variant of the PLASTIC approach that learns models of prior teammates and selects which models best predict its current teammates. An overview of this approach is given in Figure 5 and the specification of the LearnAboutPriorTeammate, SelectAction, and UpdateBeliefs functions are given in Algorithm 3. These functions are described in depth in the remainder of this section.

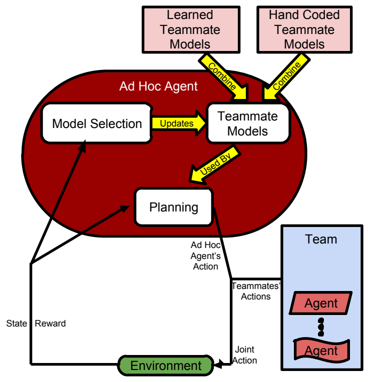


Figure 5: Overview of using the model-based approach of PLASTIC–Model to cooperate with unknown teammates.

4.2.1. *Model Selection*

In Algorithm 3, it is also necessary to select from a set of possible teammate models using SelectAction. Performing the simulations for the Monte Carlo rollouts or other planners requires that the ad hoc agent has a model of 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, if the ad hoc agent is given several possible models to choose from, the problem is more difficult. 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. In this context, $P(\text{model}|\text{actions})$ is the posterior probability of the model, given the observed actions, which is the value we want to determine. Fortunately, this value can be calculated using Bayes' theorem:

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

Algorithm 3 Instantiation of functions from Algorithm 2 for PLASTIC–Model.

```
1:  function UpdateBeliefs:
        inputs:
            BehaviorDistr          ▷ probability distr. over possible teammate behaviors
            $s$                                          ▷ the current environment state
            $a$                                               ▷ previously chosen action
        outputs:
            BehaviorDistr                                  ▷ updated probability distr.
        params:
            $\eta$                                    ▷ bounds the maximum allowed loss
2:      for $m \in$ BehaviorDistr do
3:          loss $= 1 - P(a|m,s)$
4:          BehaviorDistr$(m) * = (1 - \eta$loss$)$
5:      Normalize BehaviorDistr
6:      return BehaviorDistr

7:  function SelectAction:
        inputs:
            BehaviorDistr          ▷ probability distr. over possible teammate behaviors
            $s$                                          ▷ the current environment state
        outputs:
            $a$                                    ▷ the best action for the agent to take
        params:
            $p$                    ▷ an MDP planner that selects actions, such as UCT
    ▷ simulateAction is derived from the known environment model and
    ▷ sampling from BehaviorDistr (the teammate behavior distribution)
8:      $a = p(s)$
9:      return $a$

10: function LearnAboutPriorTeammate:
        inputs:
            $t$                                                    ▷ the prior teammate
        outputs:
            $m$                                       ▷ model of the teammate's behavior
        params:
            learnClassifier                             ▷ supervised learning algorithm
11:     Data $= \emptyset$
12:     repeat
13:         Collect $s, a$ for $t$
14:         Data $=$ Data $\cup \{(s,a)\}$
15:     $m$ = learnClassifier(Data)
16:     return $m$
```

where $P(\text{actions}|\text{model})$ is the probability of the ad hoc agent observing this series of its teammates' actions given that its teammates are acting using the specified model. $P(\text{model})$ is the prior probability of the model, which we assume to be uniform across the models in our experiments. This prior can be used to inject expert knowledge as to the relative frequencies of models. $P(\text{actions})$ is the probability of observing the series of the teammates' actions, which is used as a normalizing constant. If the correct model is in the given set of models, then the ad hoc agent's beliefs will converge to this model or a set of models that are not differentiable from this model.

On the other hand, if the correct model is not in the set, using Bayes rule may drop a good model's posterior probability to 0 for a single wrong prediction. [10] This update may punish generally well-performing models that make a single 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 [17]. Since it is expected that the ad hoc agent's models are not perfect, the agent updates its beliefs using the polynomial weights algorithm:

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

where $\eta \leq 0.5$ is a parameter that bounds the maximum loss, where higher values converge more quickly. This scheme ensures that good models are not prematurely removed, but it does reduce the rate of convergence. In practice, this scheme performs very well as the observed examples of the teammates may be arbitrarily unrepresentative of the agent's overall decision function.

If the true teammate model is in the set of models, the polynomial weights algorithm is guaranteed to converge to the true teammate model [17]. However, in the general case, the ad hoc agent has not previously seen the teammate and therefore does not have the true model of the teammate in its set of models. In this case, we expect that the more forgiving update used in the polynomial weights algorithm will perform better than a pure Bayesian update; our informal tests supported this belief.

4.2.2. Planning

This section describes the SelectAction function in Algorithm 3. When an ad hoc agent has a model of both the environment and its teammates, it can use this model to plan about the effects of its actions and how it should adapt to its teammates. Formally, the ad hoc agent can calculate $P(s'|s, a^*)$, where s' is the state resulting from the ad hoc agent taking action a^* from state s.

In this article, rather than completely calculating out this probability, the ad hoc agent uses a sample based planner to approximate this distribution. Specifically, the ad hoc agent uses UCT to quickly determine the effects of its actions and plan a sequence of actions that will be most beneficial to the team. UCT is used due to its speed and

[10] $P(\text{model}|\text{action}) = \dfrac{P(\text{action}|\text{model})P(\text{model})}{P(\text{action})} = \dfrac{0 \cdot P(\text{model})}{P(\text{action})} = 0$

ability to handle large action and state spaces, allowing it to scale to large numbers of teammates in complex domains. The modified version of the UCT algorithm that is used in this article is explained in Appendix A.2. Other planning algorithms such as Value Iteration (VI) or other approximate planners can also be used, but UCT is chosen here as it shows good empirical performance in many large domains [51].

Given the current belief distribution over the models, the ad hoc agent can sample teammate models for planning, choosing one model for each rollout similar to the approach adopted by Silver and Veness [60]. 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, state-action evaluations would be stored and performed separately 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.

While UCT is guaranteed to converge to the optimal policy, its convergence can be exponential in the number of rollouts. In addition, if the planning is performed with imperfect teammate models, the error of UCT's calculated policy is unbounded, as is the error of other approaches such as VI. Despite these limitations, UCT performs well in practice and is significantly more computationally tractable than VI.

4.2.3. Learning Teammate Models

This section describes how teammate models are learned in the LearnAboutPriorTeammate function of Algorithm 3. The previous sections described how the ad hoc agent can select the most accurate model and use it for planning, but they did not specify the source of these models. One option is that the ad hoc agent is given hand-coded models from human experts, as shown in Line 2 of Algorithm 2 and in Figure 5. However, there may not always be a source of these models, or the models may be imperfect. Therefore, a more general solution is for the ad hoc agent to learn the models. Learning allows the agent to gain a good set of diverse models over its lifespan, allowing better performance with arbitrary new teammates. The ad hoc agent builds models of past teammates' behaviors offline and then selects from these learned models online while cooperating with new teammates. It is expected that the past teammates are representative of the distribution of future teammates, though the future teammates have not yet been seen.

In our setting, a model of a teammate specifies the probability of the teammate taking each action from each state in the MDP. Formally, a model m of teammate i specifies $P(a_i|m, s)$ for all states s in the MDP. If the teammate's actions depend on the ad hoc agent's previous action (based on the reactivity of the teammates), the ad hoc agent's latest action should be included in the state in order to preserve the Markov property of the MDP.

PLASTIC–Model treats building teammate models as a supervised learning problem, where the goal is to predict the teammates' actions using the features extracted from the world state. In this setting, the ad hoc agent is assumed to have observed some past teammates' actions, in the form of tuples (s, a_i), where a_i is the action that teammate i from the MDP state s. The learning problem is to build a model m that captures $P(a_i|s)$ from this data.

The ad hoc agent also has a model of the domain, which specifies $P(s'|s, \langle a_1, \ldots, a_n\rangle)$, where $\langle a_1, \ldots, a_n\rangle$ is all of the agents' actions. By combining the models of its teammates m_j with the model of the domain, the ad hoc agent with index i can calculate the probability of its action a^* resulting in each next state by

$$P(s'|s, a^*) = P(s'|s, \langle a_1, \ldots, a_{i-1}, a^*, a_{i+1}, \ldots, a_n \rangle) \prod_j P(a_j|m_j, s)$$

Using this equation, the ad hoc agent can reason about the results of its actions far into the future.

In this article, our agent uses C4.5 decision trees as implemented in the Weka toolbox [37] to learn these models. Several other classifiers were tried including SVMs, naive Bayes, decision lists, and nearest neighbor approaches as well as boosted versions of these classifiers. However, in initial tests in the domains considered in this article, decision trees outperformed these methods in a combination of prediction accuracy and training time. On other domains, other learning algorithms may perform better; PLASTIC does not depend on using decision trees. All model learning is performed offline, reflecting past experience in the domain, but the ad hoc agent updates its belief over the models online.

To capture the notion that the ad hoc agent is expected to have extensive prior general domain expertise (as is assumed in the ad hoc teamwork setting), though not with the specific teammates at hand, we pre-train the ad hoc agent with observations of a pool of past teammates. We treat the observations of previous teammates as experience given to PLASTIC prior to deploying the ad hoc agent.

4.2.4. Adapting Existing Teammate Models

The previous sections discuss how an ad hoc agent should cooperate with teammates it has interacted with before as well as how the agent should cooperate with completely new teammates. However, in many cases, an ad hoc agent may have a limited amount of time to observe its current teammates before it interacts with them. In addition, it has extensive observations from past interactions with other teammates. For example, in pickup soccer, this scenario corresponds to having past experience in pickup soccer, showing up to a new game, and watching a couple minutes before joining in. This scenario fits the *transfer learning* (TL) paradigm, but requires the ability to leverage multiple sources of related data. In the ad hoc teamwork scenario, the observations of prior teammates correspond to the source data sets and observations of the current teammates form the target set.

In order to integrate transfer learning with PLASTIC–Model, only a minor change needs to be made to Algorithm 2. Specifically, after Line 4, we insert the lines:

$m = \text{LearnModelTL}(\text{PriorKnowledge}, \text{Observations}(\text{Teammates}))$
$\text{PriorKnowledge} = \text{PriorKnowledge} \cup \{m\}$

This alteration adds a new model to PriorKnowledge that is learned using transfer learning, combining the information from previous teammates as well as the limited observations of the new teammates.

In this setting, the transfer learning problem starts with observations over several past teammates. For each past teammate i, these observations take the form of $\mathbb{S}_i = (s^t, a_i^t)$, where a_i^t is the action that teammate i took from state s^t and t is a past timestep. In TL terminology, $\mathbb{S}_i$ is a source data set. When the ad hoc agent encounters a new teammate j, it observes a small number of this teammate's actions, resulting in a set $T = (s^t, a_j^t)$, called the target data set. Then, the ad hoc agent attempts to use these data sources to build the best model of teammate j that correctly predicts the probability of the teammate taking action a_j^t from state s^t: $P(a_j^t|s^t)$. The specifics of how the agent combines the source data sets with the target data set depends on the transfer learning algorithm.

We discuss one such transfer learning algorithm, TwoStageTransfer, in Section 3.2, and other possible transfer learning algorithms in Section 7.4. A common approach to the problem is to determine which source data set (previous teammate) is the most similar to the target data set (current teammate) and transfer knowledge from only this source data set, where the similarity refers to the probability of the teammates taking the same actions from the same states. In this work, we find that TwoStageTransfer outperforms the other algorithms tested due to the fact that it is designed to transfer knowledge from multiple data sources simultaneously. This capability allows TwoStageTransfer to transfer knowledge from all previous teammates it has encountered, increasing the accuracy of its model of the teammate's actions $P(a_j^t|s^t)$.

4.3. PLASTIC–Policy

In complex domains, planning algorithms such as UCT may perform poorly due to the inaccuracies of their models of the environment. In addition, planning a sufficiently effective behavior may be too computationally expensive to employ in these scenarios. Therefore, it may be desirable to directly learn a policy for acting in this environment rather than planning online. Learning a policy directly prevents the ad hoc agent from learning to exploit actions that work well in the model, but not in the real environment. Given that the policy learned will depend heavily on the teammates that the agent is cooperating with, it is desirable to learn a policy for each type of teammate. Then, the ad hoc agent will try to pick which policy best fits new teammates it encounters. The remainder of this section describes the PLASTIC–Policy algorithm that uses this approach, summarized in Figure 6. The subroutines used in PLASTIC–Policy are specified in Algorithm 4.

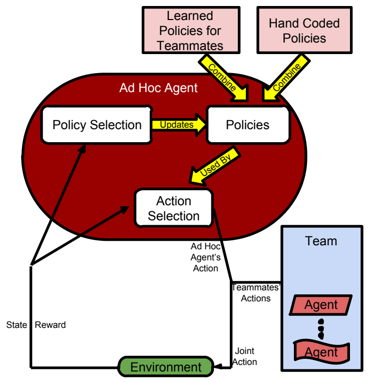


Figure 6: Overview of using the model-free approach of PLASTIC–Policy to cooperate with unknown teammates.

Algorithm 4 Instantiation of functions from Algorithm 2 for PLASTIC–Policy.

```
1:  function LearnAboutPriorTeammate:
        inputs:
            $t$                                        ▷ the prior teammate
        outputs:
            $\pi$                                      ▷ policy for cooperating with teammate $t$
            $m$                                        ▷ nearest neighbors model of the teammate's behavior
        params:
            Q-learning parameters: $\alpha$, $\lambda$, and the function approximation
2:      Data $= \emptyset$
3:      repeat
4:          Collect $s, a, r, s'$ for $t$
5:          Data $=$ Data $\cup \{(s, a, r, s')\}$
6:      Learn a policy $\pi$ for Data using Q-Learning
7:      Learn a nearest neighbors model $m$ of $t$ using Data
8:      return $(\pi, m)$

9:  function UpdateBeliefs:
        inputs:
            BehaviorDistr                              ▷ probability distr. over possible teammate behaviors
            $s$                                        ▷ the previous environment state
            $a$                                        ▷ previously chosen action
        outputs:
            BehaviorDistr                              ▷ updated probability distr.
        params:
            $\eta$                                     ▷ bounds the maximum allowed loss
10:     for $(\pi, m) \in$ BehaviorDistr do
11:         loss $= 1 - P(a|m, s)$
12:         BehaviorDistr$(m) *= (1 - \eta$loss$)$
13:     Normalize BehaviorDistr
14:     return BehaviorDistr

15: function SelectAction:
        inputs:
            BehaviorDistr                              ▷ probability distr. over possible teammate behaviors
            $s$                                        ▷ the current environment state
        outputs:
            $a$                                        ▷ the best action for the agent to take
16:     $(\pi, m) = \operatorname{argmax}$ BehaviorDistr         ▷ select most likely policy
17:     $a = \pi(s)$
18:     return $a$
```

4.3.1. Learning the Policy

PLASTIC–Policy learns about its teammates using the LearnAboutPriorTeammate function. Rather than explicitly modeling the MDP's transition function as in PLASTIC–Model, the agent directly uses samples taken from environment with its current teammates. However, online learning is sequential and can take a long time to learn a useful policy on complex domains. Therefore, it is desirable to use a distributed approach that takes advantage of the ability to run many tests simultaneously. To this end, PLASTIC–Policy performs a number of interactions in which it explores the available actions in parallel. It stores its experiences as the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state.

Using these observations, PLASTIC–Policy can learn a policy for cooperating with its teammates using existing learning algorithms. In this article, the agent uses Fitted Q Iteration (FQI) [28], as described in Appendix A.3. Alternative policy learning algorithms can be used, such as Q-learning [71] or policy search [26]. We chose to use FQI due to its good performance with continuous domains and because it does not require expert knowledge in the form of a parameterized policy.

4.3.2. Selecting Policies

This section describes the UpdateBeliefs and SelectAction functions from Algorithm 4. When an agent joins a new team, it must decide how to act with these teammates. If it has copious amounts of time, it can learn a policy for cooperating with these teammates. However, if its time is more limited, it must adapt more efficiently.

We assume that the agent has previously played with a number of different teams, and the agent learns a policy for each of these teams. When it joins a new team, the agent can then reuse the knowledge it has learned from these teams to adapt more quickly to the new team. One way of reusing this knowledge is to select from these learned policies. If the agent has previously learned a policy for cooperating with the behaviors its teammates' exhibit and it knows that its teammates are using these behaviors, the agent can directly use the learned policy. However, if it does not know which behavior types its teammates are exhibiting, the agent must select from its set of learned policies, determining which previously seen behavior best matches its teammates current behavior.

Many similar decision-making problems can be modeled as multi-armed bandit problems when the problem is stateless. In this setting, selecting an arm corresponds to playing one of the learned policies for an episode. Over time, the agent can estimate the expected values (expected chance of scoring) of each policy by selecting that policy a number of times and observing the outcome.

However, this type of learning may require a large number of trials as the outcomes of playing each policy may be very noisy depending on the complexity of the domain. Therefore, it is desirable to select from the policies more quickly. To this end, PLASTIC–Policy employs an approach based on maintaining the probability of the new team being similar to a previously observed team. At each time step, PLASTIC–Policy takes the action selected by the policy that has the highest probability of corresponding to the current teammates.

These probabilities are updated by observing the actions the team performs and using Bayes' theorem. However, Bayes' theorem may drop the posterior probability of a similar team to 0 for a single wrong prediction. Therefore, as in Section 4.2.1, PLASTIC–Policy adopts the approach of updating these probabilities using the polynomial weights algorithm from regret minimization [17]:

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

where $\eta \leq 0.5$ is a parameter that bounds the maximum loss, where higher values converge more quickly.

The learned policies do not directly give the probability of a past team taking an action. However, the experiences ($\langle s, a, r, s' \rangle$) used in learning the policies can help because they provide estimates of the teams' transition function. When the agent observes a state s and the next state s', it can update the probability of the new team being similar to each old team. For each old team, the agent finds the stored state $\hat{s}$ closest to s and its next state $\hat{s}'$. Then, for each component of the state, it computes the difference between s' and $\hat{s}'$. We assume that the MDP's noise is normal, so each difference results in a probability that it was drawn from the noise distribution. Multiplying these factors together results in a point estimate of the probability of the previous team taking the observed action.

Note that in domains in which the state has ordinal or symbolic components, different distance measures should be used. Also, while this distance function performs well in the experiments in the article, it is possible that states that are close by this measure will still be far at the behavior level, resulting in a bad estimate of the similarity. The use of the polynomial weights algorithm mitigates the effects of a small number of poor similarity estimates, but it can be overwhelmed if this measure is frequently incorrect.

5. Pursuit Results

This section presents an empirical analysis of PLASTIC's performance in the pursuit domain introduced in Section 2.4. The first tests show that PLASTIC–Model is effective for cooperating with known teammates and that UCT performs well as the planning algorithm for PLASTIC–Model. These results show that PLASTIC–Model outperforms matching its teammates' behaviors. The next tests show that PLASTIC–Model can learn to cooperate with a number of unknown teammates given prior hand-coded models of its potential teammates' behaviors for HandCodedKnowledge, and the following tests show that these hand-coded models can enable PLASTIC–Model to cooperate with a variety of previously unseen teammates.

The initial tests are used to evaluate parts of PLASTIC–Model, whether its planning is effective and whether it can select among a set of models. The next set of tests investigates the core question in this paper: whether PLASTIC–Model can reuse knowledge it learns from previous teammates in order to speed up teammates exhibiting unknown behaviors. Our first tests demonstrate that PLASTIC–Model can effectively learn models of past teammates and use these models to quickly adapt to unknown teammates. Furthermore, the results show that PLASTIC–Model is effective even when the new teammates are drawn from a substantially different set than its previous teammates. Additional tests show that TwoStageTransfer is effective for learning models of new teammates using only a small amount of observations of these teammate combined with many observations of past teammates. Using TwoStageTransfer with PLASTIC–Model allows an ad hoc agent to cooperate with a variety of unknown teammates, outperforming only reusing previously learned models.

5.1. Methods

In the pursuit domain, our ad hoc agent uses PLASTIC–Model to select its actions. To employ PLASTIC–Model, we model the pursuit domain as an MDP. States in the MDP are the current positions of all agents, and the actions are to move in one of the four cardinal directions or stay still. The transition function is deterministic except for collisions, which are handled based on a random priority assigned each time step. The reward function returns 1.0 when the prey is captured and 0 otherwise.

In Sections 5.5 and 5.6, PLASTIC–Model learns models of its teammates, as discussed in Section 4.2.3. Learning allows the agent to gain a good set of diverse models over its lifespan, allowing better performance with arbitrary new teammates. The ad hoc agent builds models of past teammates' behaviors offline and then selects from these learned models online while cooperating with new teammates. It is expected that the past teammates are representative of the distribution of future teammates, though the future teammates have not yet been seen.

To predict its past teammates' actions, the ad hoc agent uses a decision tree learning algorithm to predict the actions its teammates' would take from a given state. Rather than use a single number to represent the state, the ad hoc agent uses a factored representation. We assume that each of the resulting features is informative and that states with similar features will result in similar actions from the teammates.

Description	Num. 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 2: Features for predicting a teammate's actions. Positions are relative to the teammate.

The features in Table 2 are mostly the relative locations of other agents in the domain. The features also include whether the predator is currently neighboring the prey and whether each of the four cells around the prey are occupied by predators, which gives information about which direction the predator may move to fill the empty spots. Also, we include which of the four numbers the predator is assigned in case agents on a team are specialized based on their number. Finally, the previous two actions give a succinct, but imperfect summary of the predator's intentions; we expect that predators are likely to continue in their current direction, but the learning algorithm figures out how this history predicts the next action.

To capture the notion that the ad hoc agent is expected to have extensive prior general domain expertise (as is assumed in the ad hoc teamwork setting), though not with the specific teammates at hand, PLASTIC–Model observes a number of past teammates. Specifically, it watches teams of four predators for 50,000 steps for each past teammate type, and builds a separate model for each type of teammate. Preliminary tests show that less data can still be effective, but the focus of this research is about minimizing observations of the current teammates, not the previous ones. We treat the observations of previous teammates as experience prior to deploying the ad hoc agent. If some observations of the current teammates are available, we can improve our results using transfer learning as discussed in Section 5.6.

We evaluate how PLASTIC–Model compares to the baseline of directly copying the teammates' behaviors. Copying the teammates' behaviors tests how the team would perform if it had another teammate that matched the team rather than the ad hoc team agent. Two other possible baselines would be to have the ad hoc agent not move or select actions randomly, both of which result in the team never capturing the prey. Therefore, these baselines are not used in this article. We use the following performance metric: given 500 steps, how many times can the predators capture the prey. Whenever the prey is caught, it is randomly relocated and the predators try to capture the prey again. Results are averaged over 1,000 trials, and statistical significance is tested using a Wilcoxon signed-rank test with $p < 0.01$.

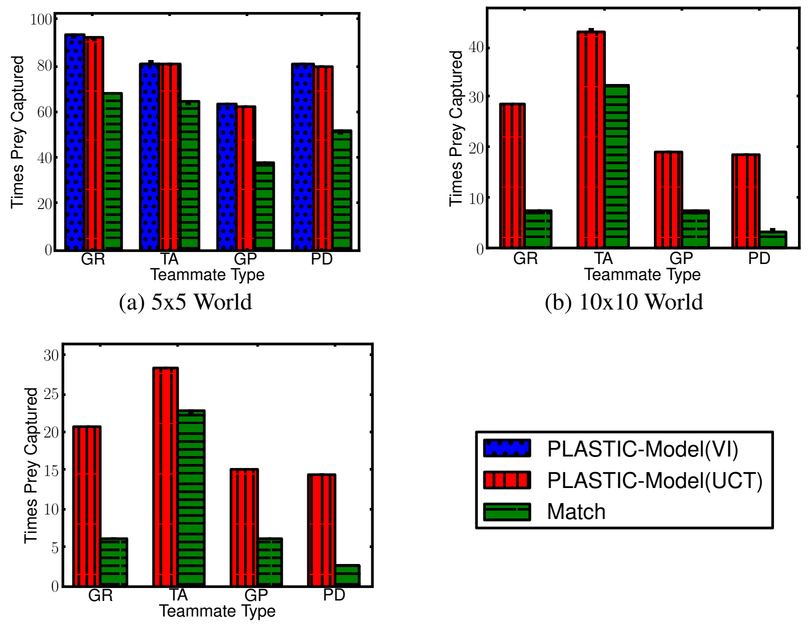
5.2. Cooperating with Known teammates

Before analyzing whether PLASTIC–Model is effective at cooperating with unknown teammates, it is first informative to test whether it can cooperate with known teammates on a known task. Specifically, we test its performance with the hand-coded teammates presented in Section 2.4.1. PLASTIC–Model is given the prior knowledge in the form of the correct hand-coded policy of its teammates behaviors for HandCodedKnowledge. Although 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. We hypothesize that PLASTIC–Model will effectively plan to deal with its known teammates and outperform matching their suboptimal behaviors.

When both the teammates and the task are known, finding the optimal behavior with PLASTIC–Model simplifies to a planning algorithm. As presented in Appendix A.1, Value Iteration (VI) is a planning algorithm that is guaranteed to compute the optimal behavior for the ad hoc agent, but it is computationally intensive to calculate. In order to scale to larger problems, it is desirable to use more efficient, approximate methods such as Upper Confidence bounds for Trees (UCT), which is discussed in Appendix A.2. Ideally, the approximate solutions will not lose too much compared to the optimal solutions. Therefore, we look at the performance of these two different planning algorithms for PLASTIC–Model, as well as the baseline of matching the teammates' behaviors.

Results for three sizes of worlds are given in Figure 7. These results show that the ad hoc agent can do much better than just copying the behavior of its teammates by using PLASTIC–Model. In the 5x5 world, following the optimal behavior found by

VI captures the prey an average of 92.82 and 81.04 times respectively when cooperating with Greedy and Teammate-aware teammates as opposed to 67.77 and 63.88 times when mimicking their behavior. The improvements of planning over mimicking the teammates increase as the worlds get larger, although VI does not scale well enough computationally to calculate the optimal behavior for these worlds. For example, on the 20x20 world, using PLASTIC–Model with UCT allows the agent to capture the prey on average 15.07 times per 500 steps when cooperating with Greedy Probabilistic teammates compared to 6.12 times when mimicking the teammates' behavior. Similarly, the agent using PLASTIC–Model captures the prey 14.47 times rather than 2.60 times when paired with Probabilistic Destinations teammates. All differences are statistically significant with $p < 0.01$.



(a) 5x5 World (b) 10x10 World

(c) 20x20 World (d)

Figure 7: Results with known hand-coded teammates.

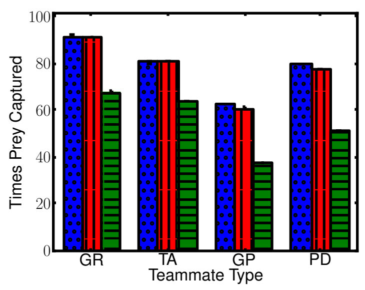
However, using the approximate planning of UCT in PLASTIC–Model is not much of a compromise, since it performs nearly as well as VI despite using much less computation time. In the 5x5 world, the agent captures the prey 91.68 and 80.45 times with Greedy and Teammate-aware agents when planning with UCT, as opposed to 92.82 and 81.043 times with VI. The difference in performance could be lowered by using more playouts in the UCT at the cost of more computation time. Given the close approximation to optimal that UCT provides, the most important difference between the methods is the time it takes to plan. On the 5x5 world, an entire UCT episode takes less than 10 seconds compared to VI's 12 hour computation (although VI only needs to run once, rather than for each episode). Furthermore, UCT is an anytime algorithm, so it can be used to handle variable time constraints and can modify its plan online as the models change. Given the good performance of UCT as well as its computational efficiency, we use it as the planning algorithm for PLASTIC–Model for the remainder of this section.

5.3. Cooperating with teammates drawn from a known set

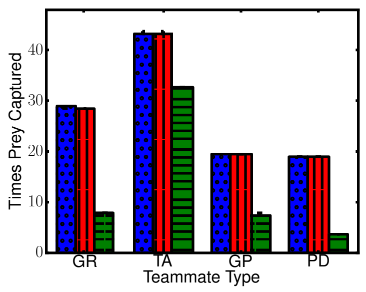
While Section 5.2 considers the case in which the ad hoc agent knows the behaviors of its teammates, the ad hoc agent may not always be this well informed. Instead, ad hoc agents will need to adapt to new teammates on the fly. Therefore, we now expand the problem, considering the case in which the ad hoc agent may encounter any of the four hand-coded predators as teammates, but it does not know which behavior its current teammates are using. The ad hoc agent does know that these teammates are drawn from the set of hand-coded predators. In other words, PLASTIC–Model receives all four hand-coded behaviors as HandCodedKnowledge and needs to determine which one best represents its teammates online. This setting is 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. These evaluations test whether PLASTIC–Model can determine which type of teammates it encounters and adapt to them. We hypothesize that PLASTIC–Model will outperform matching their behaviors and perform only marginally worse than when PLASTIC–Model knows their behaviors before interacting with them. In Sections 5.4–5.6, we explore a setting with a much larger set of possible teammates.

Ideally, assuming PLASTIC–Model has a set of possible models for its teammates as input, it should be able to determine which model is correct and plan with that model appropriately. In this setting, PLASTIC–Model uses the polynomial weights method described in Section 4.2.1 to maintain its beliefs over the teammates' types. PLASTIC–Model is given a uniform prior over the teammate types for BehaviorPrior, but PLASTIC–Model knows that the teammates are homogeneous; i.e. there were no teams with some agents following the Greedy behavior and others following the Teammate-aware behavior. The results for this scenario are displayed in Figure 8. Differentiating the deterministic teammate behaviors is straightforward because as soon as they take one action that is not expected by the deterministic behavior, the incorrect model can be removed. However, the stochastic teammate behaviors are more difficult to differentiate, as there is significant overlap in the actions that are possible for them to take.

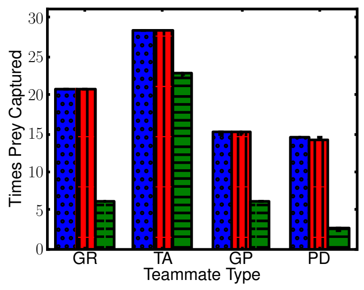
We compare PLASTIC–Model being given the four hand-coded teammate behaviors as HandCodedKnowledge to a version of PLASTIC–Model that is given only the correct model of its teammates as HandCodedKnowledge. We keep the baseline of trying to fit into the teammates' pre-designed team, denoted Match. The results are shown in Figure 8. PLASTIC–Model is statistically significantly better than Match in all scenarios. In the 5x5 world, PLASTIC–Model(All) is statistically significantly worse that PLASTIC–Model(True) for GR, GP, and PD teammates. In the 10x10 world, PLASTIC–Model(True) is significantly better than PLASTIC–Model(All) only for GR teammates, and in the 20x20 world, PLASTIC–Model(True) is significantly better than PLASTIC–Model(All) for the GR and PD teammates. These results show that PLASTIC–Model is able to quickly determine the behaviors of its teammates, losing



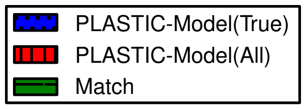
(a) 5x5 World



(b) 10x10 World



(c) 20x20 World



(d)

Figure 8: Results with unknown hand-coded teammates.

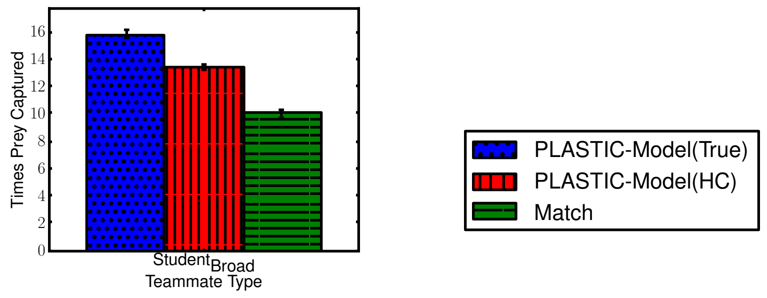
only a small amount compared to when it knows the correct teammate behavior ahead of time.

5.4. Unmodeled teammates

To this point, PLASTIC–Model has always had the benefit of having the correct model of its teammates in HandCodedKnowledge, even when HandCodedKnowledge includes incorrect models. However, PLASTIC–Model may not always be this fortunate. Therefore, we now consider the case where there are agents in A for which PLASTIC–Model does not have a correct model in HandCodedKnowledge. We again give PLASTIC–Model the four hand-coded teammate behaviors as HandCodedKnowledge, but the ad hoc agent encounters teammates not drawn from this set. To make sure we have not biased the creation of these agents towards being better for ad hoc teamwork, we used the externally-created teammates described in Section 2.4.2 as $\text{Student}_{\text{Broad}}$. PLASTIC–Model exploits similarities between the current teammates' behaviors and past teammates' behaviors, but these teammates are not explicitly designed to be similar. Nonetheless, PLASTIC–Model is able to identify and exploit similarities between teammates coming from a variety of developers.

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. For this and all following tests, we focus on the 20x20 world because it is more complex and interesting than the small worlds. We hypothesize that PLASTIC–Model will be able to determine which models best fit its teammates and use them to plan to effectively cooperate with its teammates. Our expectation is that PLASTIC–Model will outperform matching their behaviors and be outperformed by planning when their true behavior is known.

As explained in depth in Section 4.2, PLASTIC–Model maintains the probabilities of the four known models and samples from this distribution while planning. While these models are not correct, PLASTIC–Model tries to determine which of these behaviors best matches how its current teammates are behaving.



(a) (b)

Figure 9: Results with unobserved externally-created teams (Student$_{\text{Broad}}$) on a 20x20 world.

We compare 3 possible strategies for the ad hoc agent:

1. **Match** – match the teammates' behaviors
2. **PLASTIC–Model(True)** – use PLASTIC–Model with the HandCodedKnowledge initialized to the current teammates' true behavior
3. **PLASTIC–Model(HC)** – use PLASTIC–Model with the 4 hand-coded models provided as HandCodedKnowledge

Strategies 1 and 2 require the ad hoc agent to know the true behavior of its current teammates, which is not always possible. These two strategies therefore serve as baselines to compare strategy 3, which represents the true ad hoc scenario of encountering previously unseen teammates. The results in Figure 9 show that the ad hoc agents do quite well despite the incorrect models. All differences are statistically significant. For example, the PLASTIC–Model agent captures the prey 13.36 times per 500 steps rather than 9.97 times if it matched its teammate's behaviors. This result 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. While there is some loss compared to if the ad hoc agent knew the true behavior of its teammates, these 4 hand-coded models are representative enough of these externally-created teammate to achieve good results. This experiment shows that it is possible for an agent to cooperate with unknown teammates by using a set of known, representative models.

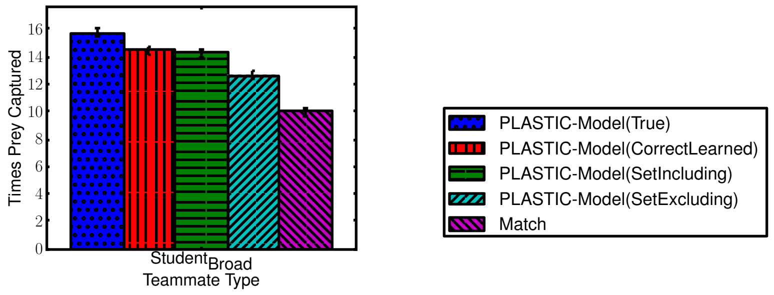
5.5. Learning About Teammates

In this section, we explore the scenario where the ad hoc agent has previously observed a number of past teammates. These past teammates are expected to be similar to the current teammates. Ideally, the ad hoc agent should be able to use the observations of its past teammates to better cooperate with its current teammates. To this end, this section evaluates how well PLASTIC–Model learns models of past teammates and then selects between these models. Specifically, PLASTIC–Model observes each past teammate for a total of 50,000 steps. This number was chosen to give the agent an excess of information for learning models as we do not care about the speed of learning. Then, PLASTIC–Model learns a decision tree to represent the behavior of each past teammate, as discussed in Section 4.2.3. This section tests the hypothesis that PLASTIC–Model can learn models of past teammates and reuse these learned models to cooperate with new teammates. The results show that this approach only marginally loses compared to knowing the teammates' true behaviors and outperforms matching their behaviors.

The teammates used in this section are those from $\text{Student}_{\text{Broad}}$, described in Section 2.4.2. These teammates are externally-created, being designed by students for a class assignment. We consider 5 behaviors for the ad hoc agent:

1. **Match** – match the teammates' behaviors
2. **PLASTIC–Model(True)** – use PLASTIC–Model with the HandCodedKnowledge initialized to the current teammates' true behavior
3. **PLASTIC–Model(CorrectLearned)** – use PLASTIC–Model with PriorTeammates being only the current teammates
4. **PLASTIC–Model(SetIncluding)** – use PLASTIC–Model with PriorTeammates including all 29 possible teammates from $\text{Student}_{\text{Broad}}$, including the current ones
5. **PLASTIC–Model(SetExcluding)** – use PLASTIC–Model with PriorTeammates including 28 possible teammates from $\text{Student}_{\text{Broad}}$, excluding the current ones

Once again, strategies 1 and 2 serve as baselines and require knowledge of the current teammates true behaviors. PLASTIC–Model(CorrectLearned) evaluates the performance of the learning algorithm, where PLASTIC–Model knows which teammates the agent is cooperating with and uses its past observations of these teammates to learn a model of them. PLASTIC–Model(SetIncluding) evaluates the more general ad hoc teamwork scenario where the current type of teammate is unknown, but the current teammates have been previously observed. Finally, PLASTIC–Model(SetExcluding) shows the true ad hoc teamwork scenario, when the ad hoc agent has never seen the current teammates, but uses PLASTIC–Model to reuse knowledge it has learned from previous teammates.



(a) (b)

Figure 10: Results with PLASTIC–Model learning models of previously observed teammates when encountering teams from $\text{Student}_{\text{Broad}}$ on a 20x20 world.

Figure 10 shows the performance of these five approaches; all differences are statistically significant. PLASTIC–Model(True) shows an unattainable level of performance as it requires perfect knowledge of the current teammates. However, learning a model by observing the current teammates does not lose too much performance, as shown by the PLASTIC–Model(CorrectLearned) line. Furthermore, having observed many teammates and needing to select from these past teammates does not generate too much loss either, as shown by the PLASTIC–Model(SetIncluding) line. Finally, PLASTIC–Model(SetExcluding) shows the performance of PLASTIC–Model when encountering a previously unseen teammate. Its performance shows that the models learned from previous teammates can do a good job of capturing the behavior of new teammates. This problem is the true ad hoc teamwork problem, when the ad hoc agent encounters teammates for which it has no prior knowledge. The gap between PLASTIC–Model(SetExcluding) and PLASTIC–Model(SetIncluding) shows that there is still room to improve for new teammates.

It is possible that the agents created by the class are biased to be similar, so all agents from $\text{Student}_{\text{Broad}}$ may share some characteristics. Therefore, we would also like to test how these learned models allow PLASTIC–Model to cooperate with teammates drawn from another set. In this scenario, we never learn models on the $\text{Student}_{\text{Selected}}$ teammates. Instead, we evaluate how well PLASTIC–Model performs when it is given $\text{Student}_{\text{Broad}}$ for PriorTeammates, but then encounters teammates from $\text{Student}_{\text{Selected}}$. Specifically, PLASTIC–Model learns 29 models, one for each teammate behavior in $\text{Student}_{\text{Broad}}$, but then encounters a 30th teammate, drawn from $\text{Student}_{\text{Selected}}$.

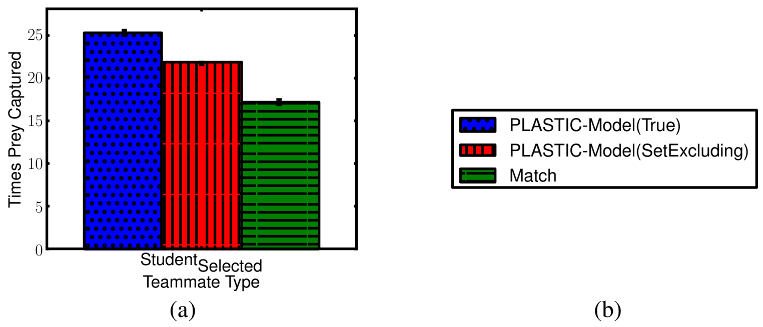


Figure 11: Results with PLASTIC–Model learning models of previously observed teammates when encountering teams from $\text{Student}_{\text{Selected}}$ on a 20x20 world.

Figure 11 gives the results of these tests, with all differences being statistically significant. Once again, PLASTIC–Model(True) shows the upper bound on performance, when PLASTIC–Model is given information about the true behavior of the teammates, which is not accessible in most scenarios. However, PLASTIC–Model(SetExcluding) performs quite well, showing that the learned models are generally useful. This approach still far outperforms matching the teammates' behaviors, despite the inaccuracies of the models. For visual comparison, videos of ad hoc agents using PLASTIC–Model to adapt to its teammates and videos of ad hoc agents using other strategies can be found online[11]

[11]`http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_Pursuit`

5.6. Learning about New Teammates

The previous section assumes that PLASTIC–Model has observed previous teammates, but not the current teammates. If instead, PLASTIC–Model observes the current teammates for a small number of steps, it can try to use this information to learn a new model about these teammates. However, given that the learning is about the current teammates, we care about the speed of learning. Therefore, PLASTIC–Model combines this information with that coming from previously observed teammates to learn a better model. Specifically, this setting permits PLASTIC–Model to use transfer learning to learn a better model of its current teammates. These evaluations test the hypothesis that using transfer learning allows PLASTIC–Model to narrow the gap between PLASTIC–Model(SetExcluding) and PLASTIC–Model(CorrectLearned) seen in the previous section.

In our tests, we assume that the ad hoc agent has previously observed 50,000 training steps of each of the past 28 teammates from $\text{Student}_{\text{Broad}}$. In addition, it has seen 100 training steps of the current teammates. Note that this is significantly less than the testing time of 500 steps, but once testing begins, PLASTIC–Model is not learning online other than adapting its belief distribution over the possible models. PLASTIC–Model could also improve its models, but we focus on evaluating the transfer learning algorithms with a fixed amount of observations of the current teammates. Both the past and current teammates in this test are taken from $\text{Student}_{\text{Broad}}$.

We evaluate four different transfer learning algorithms: TwoStageTrAdaBoost, TrAdaBoost, TrBagg, and TwoStageTransfer (discussed in Section 3.2) in PLASTIC–Model. The goal of transfer learning is to produce a model that performs well on the target data set (current teammates) by using the source data sets (past teammates). We hypothesize that TwoStageTransfer will perform the best as it explicitly reasons about the fact that the source data is coming from multiple sources. In addition, we compare these transfer learning algorithms to the performance of purely reusing the previously learned teammate models, PLASTIC–Model(SetExcluding). All of the transfer learning algorithms use decision trees as their base learning algorithm. To make the evaluations as fair as possible, for TwoStageTransfer and TwoStageTrAdaBoost, 10 different weightings were used. In TrAdaBoost and TwoStageTrAdaBoost, 10 boosting iterations were used. For TrBagg, a total of 1,000 sets were used for training classifiers, and a Naive Bayes classifier served as the fallback model. Each algorithm has some set of parameters that can be tuned, and their values were chosen in preliminary tests based on their performance and computational tractability.

Figure 12 shows the results of the four transfer learning algorithms used as subroutines of PLASTIC–Model, with all differences being statistically significant. In PLASTIC–Model, all learning of the models is performed offline with only model selection happening online during the evaluation. One baseline for comparison is if PLASTIC–Model ignores the previously observed teammates and learns a new model from just the observed 100 steps of the current teammates, shown as PLASTIC–Model($\text{CorrectLearned}_{100}$). As an upper baseline, we compare to the unattainable performance of using a version of PLASTIC–Model that observes 50,000 steps of the current teammate, shown as PLASTIC–Model($\text{CorrectLearned}_{50,000}$), which represents the best performance attainable using models learned given large amounts of data.

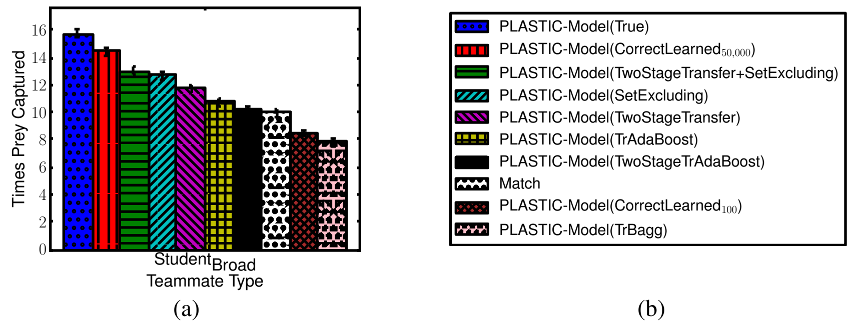


Figure 12: Comparing different transfer learning algorithms in PLASTIC–Model to improve results with ad hoc agents that have limited observations of their current teammates. Tests are in a 20x20 world with $\text{Student}_{\text{Broad}}$ teammates.

In these results, TwoStageTransfer statistically significantly outperforms the other transfer learning algorithms. In addition, combining the models learned with TwoStage-Transfer with the models learned from representative teammates in the PLASTIC–Model(TwoStageTransfer + SetExcluding) setting helps, reaching results that are statistically significantly better than PLASTIC–Model(SetExcluding). TrBagg performed poorly in this setting, mis-transferring information, possibly due to the fallback model used or the balance of target and source data. Several values of these parameters were tested, but performance remained similar.

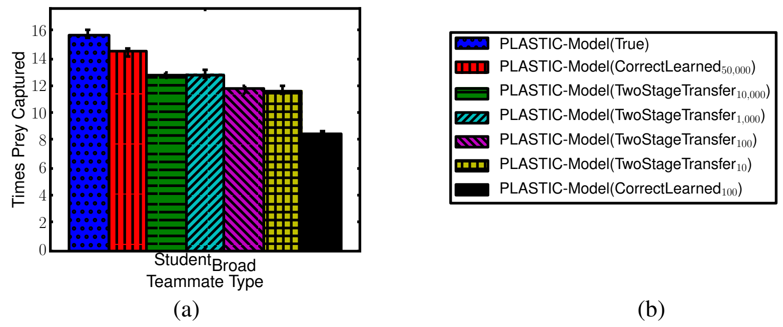


Figure 13: Evaluating PLASTIC–Model using TwoStageTransfer with varying amounts of target data. Tests are in a 20x20 world with $\text{Student}_{\text{Broad}}$ teammates.

In addition, it is important to see how much target data TwoStageTransfer needs to perform well. Therefore, we vary the order of magnitude of target data and run PLASTIC–Model(TwoStageTransfer$_i$) where i is the amount of target data provided. Figure 13 shows results with varying amounts of target data, but constant amounts of source data. The difference between the results with 1,000 steps of target data and 100 is statistically significant, but the differences between 10,000 and 1,000 or 100 and 10 are not. The results show that the performance of TwoStageTransfer does improve with more target data, but the improvement is not smooth. These results show that as few observations as 10 steps of the current teammates are sufficient for TwoStageTransfer to perform produce useful models in this scenario. We used 100 steps in Figure 12 to give other transfer learning methods enough data to perform adequately, though the results show that TwoStageTransfer still significantly outperforms them.

5.7. Summary

This section showed that PLASTIC–Model enables ad hoc team agents to cooperate with a variety of hand-coded and externally-created teammates in the pursuit domain. PLASTIC–Model gets good results when given a set of hand-coded behaviors as HandCodedKnowledge or when it has experienced a number of previous teammates as PriorTeammates. PLASTIC–Model performs well even when it has never seen the current teammates before if the ad hoc agent has experience with previous teammates that exhibit similar behaviors. Furthermore, transfer learning can learn new models quickly, allowing PLASTIC–Model to quickly adapt to new teammates. Specifically, combining models learned from TwoStageTransfer with the models of past teammates outperforms the other approaches considered. PLASTIC–Model allows the ad hoc agent to adapt to a variety of teammates. Also, the agent's behavior differs greatly when cooperating with different teammates, indicating that it is not just following a single policy with all teammates. These results show that PLASTIC–Model allows agents to reuse knowledge about previous teammates to quickly adapt by exploiting similarities to observed teammates' behaviors in the pursuit domain.

6. Half Field Offense Results

The previous section showed that PLASTIC allows ad hoc agents to cooperate with a variety of teammates in the pursuit domain given prior experiences with other teammates. However, while the pursuit domain requires a number of agents to cooperate, it is still simple compared to many realistic scenarios. Therefore, this section looks into scaling PLASTIC to a more complex domain, namely that of half field offense (HFO), described in Section 2.5. All past research on HFO has focused on creating full teams that are pre-coordinated, but this section shows that PLASTIC can handle unknown teammates without prior coordination. Given the complexity of HFO, planning using UCT requires many samples and runs into issues with imperfect modeling of the environment. Therefore, we evaluate PLASTIC–Policy in HFO. PLASTIC–Policy is more effective in this setting because it avoids the complexity of modeling the domain and teammates. Instead, PLASTIC–Policy directly learns policies for cooperating with previous teammates and then selects between these policies online for the current teammates. PLASTIC–Policy is described in depth in Section 4.3.

The results in this section show that in the HFO domain, PLASTIC–Policy is effective allowing an ad hoc team agent to reuse knowledge from past teammates to improve cooperation with its teammates. Our tests evaluate PLASTIC–Policy with teams that competed in the 2013 RoboCup 2D Simulation League and are complex, being developed over several years, which are discussed in Section 2.5.1. Despite this complexity, PLASTIC–Policy is able to learn policies for cooperating with each type of teammate, and the results show that these policies are specialized to the teammate type. Therefore, PLASTIC–Policy's approach of maintaining the probabilities of each teammate type and selecting the best policy significantly outperforms the other approaches. This section shows that PLASTIC can scale to domains with continuous states and sophisticated teammates.

6.1. Grounding the Model

Before we discuss how to learn or act in HFO, it is important to understand how we model the problem. Therefore, this section describes how we model the HFO domain as an MDP.

6.1.1. State

A state $s \in S$ describes the current positions, orientations, and velocities of the agents as well as the position and velocity of the ball. In this article, we use the noiseless versions of these values to permit for simpler learning.

6.1.2. Actions

In the 2D simulation league, agents act by selecting whether to dash, turn, or kick and specify values such as the power and angle to kick at. Combining these actions to accomplish the desired results is a difficult problem. Therefore, this article builds on the code release by Helios [3]. This code release provides a number of high level actions, such as passing, shooting, or moving to a specified point.

We use 6 high level actions when the agent has the ball:

1. Shoot – shoot the ball at the goal, avoiding any opponents
2. Short dribble – dribble the ball while maintaining control
3. Long dribble – kick ball and chase it
4. Pass_0 – pass to teammate 0
5. Pass_1 – pass to teammate 1
6. Pass_2 – pass to teammate 2

Each action considers a number of possible movements of the ball and evaluates their effectiveness given the locations of the agent's opponents and teammates. Each action therefore represents a number of possible actions that are reduced to discrete actions using the agent2d evaluation function. While using these high level actions restricts the possibilities that the agent can take, it also enables the agent to learn more quickly and prune out ineffective actions, allowing it to select more intelligent actions with fewer samples.

Additionally, the agent can select how it moves when it is away from the ball. As the agent can take a continuous turn action or a continuous dash action every time step, it is helpful to again use a set of high level actions, in this case 7:

1. Stay in the current position
2. Move towards the ball
3. Move towards the opposing goal
4. Move towards the nearest teammate
5. Move away from the nearest teammate
6. Move towards the nearest opponent
7. Move away from the nearest opponent

These actions provide the agent a number of possible actions that adapt to its changing environment, while constraining the number of possible actions.

6.1.3. Transition and Reward Functions

The transition function is defined by a combination of the simulated physics of the domain as well as the actions selected by the other agents. The agent does not directly model this function; instead, it stores samples observed from played games as described in Section 6.2.

The reward function is 1,000 when the offense wins, -1,000 when the defense wins, and -1 per each time step taken in the episode. The value of 1,000 is chosen to be greater than the effects of step rewards over the whole episode, but not so great as to completely outweigh these effects. Other values were tested with similar results.

6.2. Methods

PLASTIC–Policy learns policies for cooperating with each previously encountered team. In this article, we use Fitted Q Iteration (FQI), introduced by Ernst et al. [28]. We treat an action as going from when an agent has possession of the ball until the action ends, another agent holds the ball, or the episode has ended. Given that we only control a single agent, the teammates follow their own policies. The agent collects data about its actions and those of its teammates' in the form $\langle s, a, r, s' \rangle$ where the a is our agent's actions. The agent does not directly store the actions of its teammates, instead storing the resulting world states, which include the effects of its teammates' actions. If we controlled all of the agents, we would also consider the action from the teammates' perspectives. The agent observes 100,000 episodes of HFO with each type of teammate. These episodes contain the agent's actions when the agent has the ball as well as when it is away from the ball.

There are many ways to represent the state of a game of half field offense. Ideally, we want a compact representation that allows the agent to learn quickly by generalizing its knowledge about a state to similar states without over-constraining the policy. Therefore, we select 20 features given that there are 3 teammates:

- X position – the agent's x position on the field
- Y position – the agent's y position on the field
- Orientation – the direction that the agent is facing
- Goal opening angle – the size of the largest open angle of the agent to the goal, shown as θ_g in Figure 14
- Teammate i's goal opening angle – the teammate's goal opening angle
- Distance to opponent – distance to the closest opponent
- Distance from teammate i to opponent – the distance from the teammate to the closest opponent
- Pass opening angle i – the open angle available to pass to the teammate, shown as θ_p in Figure 14

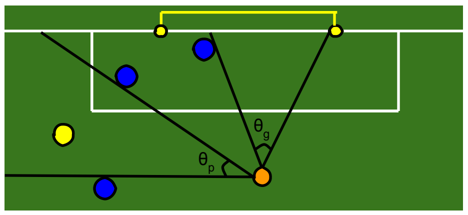


Figure 14: Open angle from ball to the goal avoiding the blue goalie and the open angle from the ball to the yellow teammate.

6.3. Evaluation Setup

Results are averaged over 1,000 trials, each consisting of a series of games of half field offense. In each trial, the agent is placed on a team randomly selected from the 7 teams described in Section 2.5.1. Performance is measured by the fraction of the time that the resulting team scores.

In this article, we use two variations on the HFO task: 1) the *limited version* with two offensive players attempting to score on two defenders (including the goalie) and 2) the *full version* with four attackers attempting to score on five defenders. In order to run some existing teams used in the RoboCup competition, it is necessary to field the entire 11 player team for the agents to behave correctly. Therefore, it is necessary to create the entire team and then constrain the additional players to stay away from play, only using the agents needed for half field offense. This approach may alter the behavior of the players used in the HFO, but our initial tests suggested that the resulting teams still perform well on the task. We choose a fixed set of player numbers for the teammates, based on which player numbers tended to play offensive positions in observed play. The defensive players use the behavior created by Helios in the limited version of HFO. In the full HFO, the defense uses the agent2d behavior provided in the code release by Helios [3].

We compare several strategies for selecting from the policies learned by playing with previously encountered teammates. The performance is bounded above by the Correct Policy line, where the agent knows its teammate's behavior type and therefore which policy to use. The lower bound on performance is given by the Random Policy line, where the agent randomly selects which policy to use. The Combined Policy line shows the performance if the agent learns a single policy using the data collected from all possible teammates, representing what an agent might do if treating this as a single agent learning problem instead of an ad hoc teamwork problem. Other options for baselines would be to have the ad hoc agent either not move or select actions randomly. However, in this setting, these behaviors result in the offense almost never scoring. Thus, these baselines are excluded from the results.

We then compare two more intelligent methods for selecting models, as described in Section 4.3.2. Specifically, our agent must decide which of the 7 policies to follow as it does not know its new teammate's behavior type. The Bandit line represents PLASTIC–Policy that uses an ϵ-greedy bandit algorithm to select policies. Other bandit algorithms were tested as were other values of ϵ, but ϵ-greedy with $\epsilon = 0.1$ linearly decreasing to 0 over the length of the trial outperformed these other methods. The PLASTIC–Policy line shows the performance of our approach, using loss-bounded Bayesian updates to maintain probabilities over which previously learned policy to use. We set $\eta = 0.1$ for updating the probabilities of the models in Equation 2. We model the noise in predicting actions using a normal distribution. This noise affects the loss function by controlling the probability function $P(\text{actions}|\text{model})$. For differences in distance predictions, we use $\sigma = 4.0$, and, for orientation differences, we use $\sigma = 40°$.

6.4. Limited Half Field Offense

Our first set of results are in the limited version of the HFO game which uses 2 offensive players competing against 2 defenders (including the goalie). Therefore, the agent only needs to adapt to a single teammate. This limited version of the problem reduces the number of state features to 8 and the number of actions while holding the ball to 4, while the number of actions away from the ball stays at 7. These evaluations test the hypothesis that PLASTIC–Policy can quickly converge to selecting the best policy, losing only a small amount compared to the correct policy. In addition, we hypothesize that PLASTIC–Policy will converge much faster than the bandit-based approach and will also outperform combining the data from all of the agents to learn a single, combined policy. The results are shown in Figure 15, with the error bars showing the standard error.

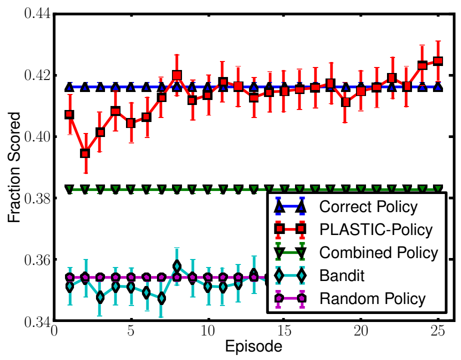


Figure 15: Scoring frequency in the limited half field offense task.

The difference between the Correct Policy and Random Policy lines shows that selecting the correct policy to use is important for the agent to adapt to its teammates. The gap between the Correct Policy and Combined Policy shows that knowing the correct teammate is better that grouping all teammates together. While the Bandit line does not show much learning in Figure 15, it does continue learning over time. Its performance converges to scoring 0.418 of the time after approximately 10,000 episodes, though it does score approximately equal to the combined policy (0.382) after 1,750 episodes. Its slow speed is due to the fact that its observations are noisy estimates of the policies' effectiveness and are only received after each game of HFO. In addition, the scoring fractions of the different strategies are fairly close, so determining the best one given the amount of noise is difficult.

On the other hand, the PLASTIC–Policy line shows fast improvement, converging to the performance of the Correct Policy line. This quick adaptation is due to two factors: 1) the better estimations of which policy fits the teammates and 2) the frequency of the updates. The estimations of the probabilities are better as they measure how each agent moves, rather than only using a noisy estimate of how the policy performs. The updates are performed after every action rather than after each episode; so updates are much more frequent. These two factors combine to result in fast adaptation to new teammates using PLASTIC–Policy. The differences between the performance of PLASTIC–Policy and Combined Policy and Bandit are statistically significant using a two population binomial test with $p < 0.01$ for all episodes shown in Figure 15. Note that while PLASTIC–Policy outperforms Correct Policy in the final episodes, this difference is due to noise and is not statistically significant. Videos of the performance of PLASTIC–Policy compared to other strategies can be viewed online.[12]

To understand the learning of PLASTIC–Policy, it is useful to look at its beliefs, shown in Figure 16. This graph shows the probability of the correct model of the current teammates as well as the probability that correct model has the highest probability (with ties contributing a probability of $\frac{1}{\#tied}$). While the probability of the correct model takes over 15 episodes to reach above 90% probability, the correct model becomes the maximal model 90% of the time after just 5 episodes. This result explains why taking the maximal model gives such good performance in PLASTIC–Policy. Note that choosing the maximal model does not create premature convergence because each action the teammates take allows PLASTIC–Policy to update the probability of those teammates.

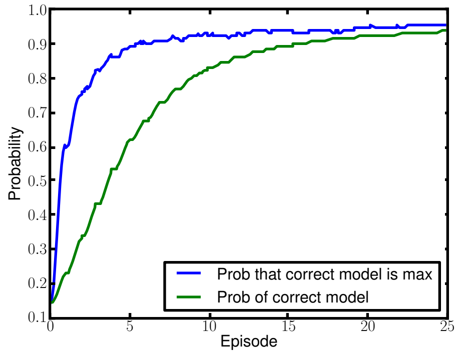


Figure 16: Belief of the probability of the correct model ($P(m^*|s,a)$) and probability of the correct model having the highest probability ($P(p^* = \max p_i|s,a)$) calculated by PLASTIC–Policy in the limited HFO task.

In summary, the results in this section show that PLASTIC–Policy is effective for cooperating with unknown teammates on a complex domain with continuous state and continuous actions. PLASTIC–Policy is able to learn policies for cooperating with previous teammates and quickly select from these policies to efficiently cooperate with new teammates.

6.5. Full Half Field Offense

Our second set of results are in the full HFO game with 4 offensive players versus 5 defenders (including the goalie). In this setting, our agent needs to adapt to its three teammates to score against the five defenders. This setting tests the hypothesis that PLASTIC–Policy can learn intelligent policies for cooperating with its three teammates and quickly select between these policies when cooperating with unknown teammates. We expect that PLASTIC–Policy will outperform selecting policies using a bandit-based approach or learning a single policy to cooperate with all teammates. In addition,

[12] `http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO`

we hypothesize that PLASTIC–Policy will only lose marginally compared to the gold standard of knowing the best policy before interacting with its teammates.

The results for this setting are shown in Figure 17. As in Section 6.4, the upper bound on performance is given by Correct Policy and the lower bound is given by Random Policy. The Bandit setting learns slowly, reaching a performance of 0.357 after approximately 20,000 episodes. It outperforms the combined policy (0.350) after 12,000 episodes. Once again, PLASTIC–Policy quickly converges to the correct policy's performance, outperforming the Bandit and Combined lines. These results show that PLASTIC–Policy quickly learns to cooperate with unknown teammates. Using a two population binomial test with $p < 0.05$, PLASTIC–Policy's performance is statistically significantly better than Combined Policy and Bandit from episode 3 on. For visual comparison, videos of ad hoc agents using PLASTIC–Policy and other strategies to cooperate with its teammates can be viewed online.[13]

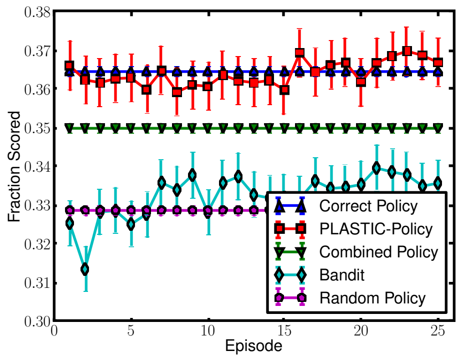


Figure 17: Scoring frequency in the full half field offense task.

We again look at PLASTIC–Policy's beliefs over time in Figure 18. In this figure, we can see that PLASTIC–Policy takes several episodes to be convinced that the correct model is the true model of the current teammates because of the noise of the whole team's actions. However, greedily selecting the highest probability model's corresponding policy performs well because the correct model is maximal 90% of the time after just 5 episodes. This result shows that PLASTIC–Policy learns quickly and can take advantage of its continuing exploration of its teammates despite only selecting what it believes in the current best policy.

6.6. Summary

The results in this section demonstrate that PLASTIC–Policy can scale to complex domains requiring coordinating with many teammates, continuous states, and continuous actions. PLASTIC–Policy can efficiently select good policies for cooperating with its current teammates from a set of policies learned for cooperating with past teammates. Playing any one of these policies would perform poorly across the spread of

[13]`http://www.cs.utexas.edu/~larg/index.php/Ad_Hoc_Teamwork:_HFO`

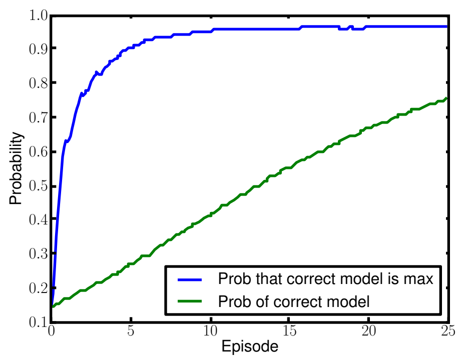


Figure 18: Belief of the probability of the correct model ($P(m^*|s,a)$) and probability of the correct model having the highest probability ($P(p^* = \max p_i|s,a)$) by PLASTIC–Policy in the full HFO task.

possible teammates. However, PLASTIC–Policy is able to quickly select a good policy for the specific teammates with which it is cooperating.

7. Related Work

While the preceding sections discuss the research problem and the approach used to tackle that problem, this section focuses instead on the problem of situating this article in literature. We first provide overviews of some areas that are closely related to ad hoc teamwork, beginning with general multiagent coordination research and then moving on to research into opponent modeling. We follow this with a discussion of selected works that use the domains discussed in this article. Finally, we conclude with a discussion of current research on ad hoc teamwork.

While there is a large amount of work related to this research, this article brings many new ideas to the table. With respect to ad hoc teamwork, this article moves ad hoc teamwork to an empirical setting and tackles more complex problems than those studied previously. Rather than requiring shared communication and coordination protocols like past research on multi-agent teams, this work describes agents that can cooperate without these shared protocols. Compared to opponent modeling, this work creates agents that adapt more quickly to other agents at the price of making the stronger assumption that they are cooperating towards a shared goal. Furthermore, this article expands the formulation of the pursuit and RoboCup domains to include teammates that come from a variety of sources that need to learn to cooperate on the fly.

7.1. Multi-Agent Teams

A large body of research on coordinating multi-agent teams exists, specifying standardized protocols for communication and shared algorithms for coordination. One such algorithm is STEAM [68], in which team members build up a partial hierarchy of joint actions. In addition, agents monitor the progress of their plans, adapting their plans when conditions change. STEAM is designed communicate selectively, reducing the amount of communication required to coordinate the team. STEAM has been shown to be effective in a number of domains, ranging from simulated RoboCup soccer domains to controlling a number of autonomous helicopters for attack scenarios.

Another approach to coordinating multi-agent teams is Generalized Partial Global Planning (GPGP) [25]. GPGP works with heterogeneous teams on a variety of tasks and allows for a number of modular coordination mechanisms. Decker and Lesser find that different coordination mechanisms perform better for different tasks and endorse the idea that there should be a library of coordination mechanisms to choose from for new tasks.

STEAM and GPGP expect that agents know the team's goal as well as the tasks required to accomplish the goal. However, another line of research explores domains in which this information is not known; instead, agents must learn the tasks required to accomplish their goals. An example of this approach is Lauer and Riedmiller's work which introduces Distributed Q-learning [53] for learning in multiagent, fully cooperative settings. The authors model the problem as a multiagent MDP and adopt a model free approach. In Distributed Q-learning, each agent maintains a local policy and Q-value function that depends only on its own actions. Using this approach, the agents' policies converge to the optimal joint-actions in deterministic domains.

However, all of these approaches expect that the entire team shares the same coordination algorithm. PLASTIC does not require any shared communication or coordination protocols and does not assume that the teammates are necessarily adapting to the ad hoc team agent.

7.2. *Opponent Modeling*

The work discussed in the previous section assumes that the whole team is cooperative, trying to accomplish a shared goal. Another scenario that may occur in multiagent teams is the fully competitive setting, where agents attempt to achieve mutually exclusive goals. Opponent modeling research explores this problem, explicitly modeling the other agents in the domain. While research into cooperative teams appears to be more similar to ad hoc teamwork given that agents are trying to accomplish shared goals in both settings, opponent modeling is often more similar to ad hoc teamwork. This similarity stems from the importance of understanding and reasoning about the other agents in the domain in opponent modeling, which is also necessary for robust ad hoc team agents. Rather than trying to represent the entire field of opponent modeling, this section summarizes some of the major lines of inquiry into the problem that are relevant to this article.

One such line of inquiry is theoretically motivated, exploring what can be proven about interacting with opponents. An algorithm that shows this reasoning is the AWESOME algorithm [23]. AWESOME is a learning algorithm for repeated normal form games. When it plays against opponents that use the same algorithm, the AWESOME agents will converge to playing the Nash equilibrium, the optimal behavior if all agents are rational. When playing against stationary opponents, an AWESOME agent learns to exploit them optimally. These results show that the same algorithm can exploit simple opponents while still not getting exploited by smart agents. In the same vein of theoretical analysis, Chakraborty and Stone developed the CMLeS algorithm [21]. CMLeS extends to exploiting memory-bounded teammates while retaining the convergence to the Nash equilibrium in self play. CMLeS reverts to playing a maximin strategy when playing adversaries that are not memory-bounded, retaining good payoffs in the worst case scenario.

Another exciting line of opponent modeling research is into using game theory to solve real world security problems. For example, Korzhyk et al. [52] discuss the use of the Stackelberg game model to design intelligent strategies. This work has been applied for deploying security at the LAX airport as well as scheduling the US Federal Air Marshals. In Stackelberg games, the leaders act first, and their opponents can observe these actions before responding. The solution to this problem is robust to the opponents' actions, minimizing the risk. This line of research shows that game theoretic approaches can be applied to real world problems with great effect, minimizing the resources required to protect a resource while maximizing its safety. In this research, the authors generally assume that other agents are opponents and act optimally, while in ad hoc teamwork the other agents are teammates and may not act optimally.

One avenue of research that combines theoretical analysis with empirical results is in the area of computer poker. For example, Bard et al. [10] look at how to adapt to an opponent's strategy in heads-up limit Texas hold'em poker. The authors approximate the Nash equilibrium strategy through the use of counterfactual regret (CFR) [78], which limits the amount that their agent can be exploited. To adapt to weaknesses in other players while remaining robust to being exploited, Bard et al. learn a portfolio of counter strategies that are limited best responses to expected opponents' behaviors.

7.3. Experimental Domains

Isaacs performed seminal research on pursuit and evasion [44], and the problem was further explored by Benda et al. [15]. Benda et al. explore varying predators' ability to communicate, even considering a central strategy, but communication carries a cost. In this setting, communication may be limited or may lower the amount of computation time taken per step. More research into the range of research possibilities in the pursuit domain is explained in the survey paper by Stone and Veloso [66].

Most previous research focused on developing coordinating the predators before deploying them, rather than learning to adapt to unseen teammates. For example, MAPS [70] considers the pursuit problem in dynamic and partially observable environments. In this work, the prey intelligently plans its actions to avoid capture. MAPS uses two coordination strategies to position the predators in locations to cover possible escape directions of the prey.

Ishiwaka et al. [45] focus on the pursuit task in which all of the predators are learning online, using reinforcement learning. The predators start as homogeneous agents, but diverge during the learning process. This work considers the pursuit task with partial observability in a continuous domain.

In addition to the research performed in the pursuit domain, there has also been substantial amounts of research in RoboCup, including walking, positioning, and vision. However, the most relevant research from RoboCup is in the area of team coordination.

One early exploration of learning in the RoboCup simulation domain was performed by Stone [62]. This book describes a flexible team structure as well as a novel learning approach called layered learning. In particular, it introduces the concept of a "locker-room agreement." Locker-room agreements are pre-determined multiagent protocols that define the flexible teamwork structure and the inter-agent communication protocols. However, this work relies on the entire team sharing this locker-room agreement, which cannot be assumed in ad hoc teamwork.

Another aspect of research in RoboCup is how to characterize the teams' behaviors in these complex multiagent systems. Almeida et al. [9] explore this problem in the RoboCup 2D simulated robotic soccer league, using logs of play as their input information. The authors explore the complexity of team's behaviors as well as discovering guidelines for creating new plans for teamwork. However, the proposed approach requires a substantial amount of observations of the team, which is not usually available in ad hoc teamwork scenarios.

One set of online adaptations to other agents is from the Small Size League (SSL) of RoboCup. Biswas et al. [16] explore how to plan about opponents' strategies. Their approach attempts to pull opponents out of position, leaving openings that their strategy can then exploit. In addition, they detect potential threats based on the positions of opponents and adapt to defend these threats. These online adaptations show that agents can adapt to other agents in the domain on the fly, in just a single game. While this general approach could be applied to other multiagent settings, the current version of it relies on strong assumptions about the domain.

RoboCup encourages a substantial amount of research into multiagent systems in complex domains. The majority of this research focuses on coordinated teams of agents and is thus not directly applicable to ad hoc teamwork.

7.4. Transfer Learning

In Section 3.2, we discuss the transfer learning algorithm, TwoStageTransfer [14], that we use for transferring knowledge from many past teammates to the current ones. However, there are several other transfer learning algorithms that can be used. TrAdaBoost [24] is a boosting-based algorithm, in which the source and target data sets are lumped together and then a model is learned via boosting. TwoStageTrAdaBoost [57] was designed in response to problems of TrAdaBoost overfitting the training data. Therefore, TwoStageTrAdaBoost first searches over a set of possible weightings of the source data points, and determines which weighting is best using cross-validation. While the other transfer learning algorithms described here focus on using boosting, bagging approaches have also shown promise, specifically in the form of TrBagg [50]. The TrBagg algorithm uses bootstrap sampling to create a number of data sets taken from the combined source and target data sets. Then, a model is learned on each data set, and these models then undergo a filtering phase, using cross validation to determine which models are most helpful.

There has also been some research into transferring knowledge from multiple sources. Yao and Doretto [74] introduced two transfer learning algorithms for handling multiple sources. The first, MultiSourceTrAdaBoost, uses an instance-transfer approach, reusing data from the source tasks for training the target classifier. Alternatively, TaskTrAdaBoost employs a parameter-transfer approach where it is assumed that the target data shares some parameters with some of the source data sets. Another approach for transfer learning with multiple source sets is the work of Huang et al. [42]. They propose the SharedBoost algorithm to select the best features for prediction from a small number of source data sets for text classification. Zhuang et al. [77] investigate using autoencoders to determine a feature mapping that allows them to train multiple classifiers from the source domain and apply them effectively on the target domain. Similarly, Fang et al. [29] introduce an algorithm that determines a shared subspace among the labels for multiple sources, where each sample is given multiple labels. Then, this subspace is used to transfer knowledge to the target domain. Another approach was developed by Ge et al. [30]. The authors introduce an online algorithm that transfers knowledge from multiple sources in a principled way, achieving a no-regret guarantee compared to the offline algorithm. These algorithms provide a promising step towards effectively handling multiple sources. Further progress on transfer learning from multiple sources may lead to exciting advances in the area of ad hoc teamwork and can be incorporated into the learning component of PLASTIC.

7.5. Ad Hoc Teamwork

A significant portion of research on ad hoc teamwork takes a theoretical approach to the problem. These approaches focus on simple settings such as the bandit domain or matrix games and try to prove the optimality of their approach under certain conditions. Other researchers focus on empirical approaches, showing that their algorithms apply to ad hoc teamwork problems in practice. Let us first consider theoretical contributions coming from analysis of ad hoc teamwork in the bandit domain.

One example of the theoretical approach using the bandit domain is Stone and Kraus's research [65]. In this work, the authors consider a multiagent version of the setting with a knowledgeable agent attempting to teach its novice teammate. This work differs from existing teaching literature in that the teacher is embedded in the domain, so its teaching actions have an explicit cost. This line of research proves that the ad hoc agent acting as teacher can optimally lead its teammate to achieve the best team payoff. These papers differ from the work in this article in that they assume that the novice teammate's policy is known (TeamK $= 1$) and their results are only directly applicable to the bandit domain, while we consider domains that are too large to be proven to be tractable in a similar fashion.

While the bandit domain allows for multiagent research, the majority of work on it is single agent and therefore not focusing on the ad hoc teamwork problem. A more common domain for looking at interactions between agents is in matrix games, where agents act simultaneously and receive rewards. This domain allows for multiagent interactions, but remains simple enough for theoretical analysis.

An early paper that looks into ad hoc teamwork in matrix games was that of Brafman and Tennenholz [19]. In their paper, they investigate agents performing a repeated joint task, where one agent attempts to teach a novice agent. The authors could only affect the ad hoc agent, i.e. the agent acting as a teacher. In this work, they use a game-theoretic framework and only consider teammates that either maximize their expected utility or use reinforcement learning. Overall, they consider a number of strategies for the agent to play, including a reinforcement learning agent as well as some well established hand-coded policies.

Building on this idea, Stone et al. [64] investigate ad hoc teamwork in matrix games with a theoretical focus. They explore how an ad hoc agent should cooperate with a best response teammate while maximizing the team's shared rewards. Best response agents choose the action that gives the highest payoff assuming that its teammates continue playing their last observed action. In this work, the ad hoc agent knows the payoff matrix as well as the teammate's behavior (TeamK $= 1$), so the difficulty is to plan the optimal path to lead the best response teammate to the best payoff. This work was expanded by Agmon and Stone [2] to include more than one teammate. Agmon and Stone show that the best payoff is not always reachable when the team is larger than two agents, but they come up with a way of describing the optimal team payoff as the optimal steady cycle and show how to lead a team to that cycle. This work was further expanded to the case where the teammates' behaviors are not fully known [1] (TeamK < 1), instead assuming that the ad hoc agent knows that its teammates are using a behavior from a known set. The authors describe an algorithm, REACT, that balances the potential costs of different assumptions about the teammates' behaviors and show that REACT empirically performs well on a large number of matrices. This line of research differs from that of this article due to its focus on theoretical analysis, which limits the work to the simple matrix game setting. In addition, this article considers a wider range of possible teammate behaviors as well as cases where the ad hoc agent has less knowledge of its teammates.

Stone et al. [64] and Agmon and Stone [2] both assume that the teammates' behaviors are known, though Agmon et al. [1] relax this assumption, instead assuming that the teammates' behaviors are drawn from a known set. Chakraborty and Stone [22] further relax this knowledge of the teammates' behaviors in ad hoc teamwork scenarios in matrix games. This work extends earlier work by Chakraborty and Stone [21] for opponent modeling. The authors propose a new algorithm, LCM, that tries to achieve optimal performance with teammates that use a limited features derived from the history. With other teammates, LCM ensures that the team receives the security value of the matrix game. LCM does this by determining which features best explain its teammate's behavior, and, if no set of features explains its behavior, LCM reverts to playing the safety strategy. This approach performs optimally with some teammates, but this form of learning takes substantially longer than PLASTIC. Unlike PLASTIC, LCM does guarantee a safety value with any teammates, but in practice this safety value is often low compared to what the team could receive and ensuring a safety value in more complex tasks requiring coordination is often impossible.

While matrix games serve as a good testbed for looking at interactions between agents in ad hoc teamwork scenarios, they are limited to stateless interactions. Wu et al. [72] scale theoretical analysis of ad hoc teamwork to some more complex, though still theoretically tractable, domains. In this work, the authors investigate ad hoc teamwork with few assumptions about the behaviors of the teammates. Their ad hoc agent plans using MCTS and uses biased adaptive play to predict the actions of teammates. Biased adaptive play can be used to estimate the policies of teammates from their previous actions. They test their agent on three domains: cooperative box pushing, meeting in a 3×3 grid, and multi-channel broadcast. They consider the case where the ad hoc agent knows the environment, but not its teammates. These teammates are referred to as unknown teammates (UTM), and two types of teammates are used in each domain: UTM-1 agents that follow a fixed set of actions and UTM-2 agents that try to play the optimal behavior but have partial observations. Their work shows that their approach can adapt to these teammates to accomplish their tasks. While this work explores several domains, all of the domains used are fairly simple. Additionally, their ad hoc agent is given a large amount of expert knowledge and the set of possible teammates is limited compared to this article.

Another approach to scaling ad hoc teamwork beyond only matrix games is the work of Albrecht and Ramamoorthy [4, 5], though they also consider matrix games in their work. In this work, they consider the case where the ad hoc agent is given a set of possible types of its teammates and introduce a new formal model to represent this problem. In their setting, the problem is for the ad hoc agent to determine the type of its teammates. Their approach (HBA) combines the idea of Bayesian Nash equilibria with the Bellman optimality equation. HBA maintains the probability of each of the provided teammate types and maximizes its expected payoffs according to the Bellman principle. In later research [6], Albrecht and Ramamoorthy explore the convergence bounds of HBA. Specifically, they prove convergence bounds when an ad hoc agent knows its teammates are drawn from a known set and consider how accurate the expert-provided types need to be for HBA to solve its task. This line of research is closely related to that of this article, but differs in some notable ways. Given the similarity of HBA and PLASTIC, we expect that much of their analysis could be generalized to PLASTIC. However, HBA assumes that the agent is provided with expert knowledge about possible teammates. In this article, we show that PLASTIC can learn from previous teammates and can apply these imperfect learned models to cooperate with new teammates. Furthermore, we show that PLASTIC scales to more complex domains than those used to evaluate HBA. Specifically, reusing precalculated policies to adapt to new teammates allows PLASTIC–Policy to handle substantially harder problems than those explored by Albrecht and Ramamoorthy.

While theoretical analysis of problems can create exciting new algorithms for ad hoc teamwork, an important question is how well these algorithms fare in more complex empirical analyses. A work that looks at more complex coordination in ad hoc teams is that of Bowling and McCracken [18]. In the domain of robot soccer, Bowling and McCracken measure the performance of a few ad hoc agents, where each ad hoc agent is given a playbook that differs from that of its teammates. In this paper, a play refers to a team plan that specifies when the play applies, termination conditions, and roles for the players. In this domain, the teammates implicitly assign the ad hoc agent a role, and then react to it as they would any teammate. The ad hoc agent analyzes which plays work best over hundreds of games and predicts the roles that its teammates will play. This work explores a very similar setting to that used in this work, though it learns over a significantly longer time scale. However, they focus on agents that are constrained to be similarly designed but have a different playbook, rather than being designed completely independently. In addition, their approach relies on having a playbook of possible plays that specifies roles for all agents on the team. In many domains, agents may not have such a playbook, so this approach cannot be directly applied to these domains.

Jones et al. [46] also consider robotic ad hoc teams, but they expand their analysis to heterogeneous robots. The authors explore ad hoc teams operating in the treasure hunt domain and implement their algorithms on real heterogeneous robots searching new environments for treasure. The authors focus on how agents can allocate roles amongst a team in a decentralized fashion. However, they assume that the agents share a communication protocol that they use to bid on different roles in an auction as well as a shared coordination protocol for how to assign tasks given this communication. This article explores scenarios in which these shared protocols do not exist, as they may not always be present in ad hoc teamwork scenarios.

While the previous works mainly focus on small teams of agents, there is also research into how to affect large teams of agents. Specifically, researchers have investigated how to use a small number of agents under their control (ad hoc agents) to affect the behavior of flocks of agents, such as flocks of birds or fish. The ad hoc agents encourage the team to reach a specified goal. An early paper in this area was written by Han et al. [38], prior to Stone et al.'s formulation of ad hoc teamwork. This work focused on adding a "shill" agent to the flock, that corresponds to the ad hoc team agent in our terminology. This agent was designed by the authors and attempts to move the flock in a desired direction. Further research on this line includes the work of Genter et al. [32] and Genter and Stone [33]. This research proves bounds on the number of actions required to control the flock's behavior. In addition, they provide an algorithm that empirically outperforms other methods by using short-term lookahead planning. The authors expand the problem to consider multiple ad hoc agents. This line of research differs from this article in that it focuses on scenarios in which the teammates' behaviors are known (TeamK $= 1$), rather than needing to learn about teammates.

While the previous works all consider how an ad hoc agent should act to improve the performance of its team, another consideration is how to choose agents to form a new team given a much larger set of possible agents. Liemhetcharat and Veloso [54] explore this idea, selecting which agents will form a new ad hoc team. Given that different agents are better at performing different roles on the team, it is important to select agents that fill the roles in a beneficial way. In addition, there are synergies in the team, where some pairs of agents work better with each other than with other agents. These complexities lead to interesting questions into how to select teammates from this set of agents. The authors come up with a novel representation of this problem, called a synergy graph, and show how to learn this graph. While it also investigates ad hoc teamwork, this research focuses on the problem of selecting agents for an ad hoc team rather than the question explored in this article, how an agent should act on the ad hoc team.

One such way is as an Interactive POMDP (I-POMDP) [34]. I-POMDPs model adversarial interactions of agents by examining what an agent believes about the other agents and these agents' beliefs. The graphical counterparts of I-POMDPs are known as Interactive Dynamic Influence Diagrams (I-DIDs) [27]. I-DIDs provide concise representations of the beliefs about other agents and allow for nesting I-DIDs to represent these beliefs. Both I-POMDPs and I-DIDs could be used to model the problems studied in this article. However, both have issues with the potential exponential blowup of beliefs as the size of the problem grows. While work has been performed to increase the efficiency of algorithms for these models [61, 75, 76], they remain computationally intractable for the size of problems studied in this article.

I-POMDP Lite [41] is an especially relevant formulation of I-POMDPs which improves scalability by modeling other agents as nested MDPs. Current implementations of I-POMDP Lite have been shown to scale up to problems with 18,000 states. In the pursuit domain, there are approximately 10^{13} states, and in the HFO domain, the states are continuous and therefore infinite. Applying I-POMDP Lite to ad hoc teamwork is an exciting area for future research, but the current instantiations of it have not been shown to scale to the size of problems considered in this paper.

7.5.1. Dimension Analysis

For this related work in ad hoc teamwork, it may be helpful or informative to consider where these problems fall on the dimensions described in Sections 2.1. We would like to calculate the exact values of the dimensions for each of the domains as we did in Sections 2.4 and 2.5, but this calculation requires more information about the exact formulations of the domains and teammates than is typically available in the publications that are available to us. Therefore, we instead give some rough estimates of where these problems lie on the dimensions. The three dimensions we consider are team knowledge, environment knowledge, and the reactivity of teammates.

We begin by discussing the team knowledge (TeamK) of ad hoc agents in these domains, which shows how much the ad hoc agent knows about its teammates prior to cooperating with them. The majority of the related research considers cases where the teammates are known, so TeamK is 1 or close to 1. Notable exceptions of this include Liemhetcharat and Veloso's work [54] as well as Wu et al.'s work [72] which consider completely unknown teammates, where TeamK is 0. Also, Agmon et al. [1] and Albrecht and Ramamoorthy [5] assume that their teammates are drawn from a known set, so TeamK is between 0 and 1; we estimate that TeamK lies in the range [0.3,0.7] for these works. Additionally, Bowling and McCracken [18] consider situations with teammates that do not share a codebook with the ad hoc agent. We estimate that TeamK is fairly high in these settings, in the range [0.6,0.8], given that effective soccer plays are similar compared to random movement of teammates. From this analysis, we can see that most existing works focus on situations where the teammates are fairly well known; only a few consider scenarios where the teammates are initially unknown.

Let us now consider the environmental knowledge (EnvK) of these domains, where the environmental knowledge explains how much the ad hoc agent knows about the transition and reward dynamics of the domain before beginning. The majority of the works in this section assume that the ad hoc agent has full knowledge of the domain, so EnvK $= (1, 1)$ for these settings. While there may be noise in these domains, the ad hoc agent is expected to know the level of noise and therefore know the true distribution of next states. Exceptions to this include Jones et al. [46] and Liemhetcharat and Veloso [54] where the ad hoc agent does not initially know the reward function and may have limited knowledge of the transition function. We estimate that the knowledge of the transition function lies in [0.7,1.0] for these works, and the reward knowledge lies in [0,0.5]. This analysis suggests that research into ad hoc teamwork has not focused on learning about the domain. Instead, agents are assumed to know the domain and instead focus on learning about teammates and planning how to cooperate with them.

The reactivity of the teammates in these domains (Reactivity) covers a large spread of values. All of the domains assume that the teammates are at least partially reactive to the ad hoc agents, or it would not be worth considering the problem as a multiagent setting. This reactivity varies significantly based on the domain. When ideal actions are fairly obvious to teammates, interactions with the ad hoc agent are unlikely to change the teammates' behaviors, leading Reactivity to be close to 0. On the opposite end of the spectrum, when the teammates have high uncertainty about the best actions ahead of time, the ad hoc agent's actions can significantly affect their actions, leading to values of Reactivity close to 1. Given that so much research assumes that ad hoc agents know their teammates and the domain well, the majority of focus has been on how to plan to cooperate effectively with teammates. Exploring planning in ad hoc teamwork encourages researchers to investigate settings with varying amounts of teammate reactivity, as this dimension is the most influential on planning.

While calculating the exact values for each of the three dimensions (TeamK, EnvK, and Reactivity) for each domain studied in the related work would be useful, it is impossible to calculate these values without complete knowledge of the domain and teammates. Even so, these rough estimates of the dimensions for these problems lead to some interesting conclusions. Specifically, existing research has done a good job of exploring how to plan to cooperate with teammates, covering the gamut of teammate reactivity. On the other hand, ad hoc team research has focused largely on problems with high team knowledge and high environmental knowledge, with less work exploring how agents can learn about their teammates and the domain. Future work in ad hoc teamwork should address this gap and explore settings in which the ad hoc agent needs to learn more about its teammates and the domain. In real world scenarios, robots will need to be constantly adapting to their changing environments as well as new teammates they may encounter. Therefore, it is important for ad hoc teamwork research to explore these settings, where agents must reason about the tradeoff between exploring the domain, exploring their teammates, and exploiting their current knowledge.

7.5.2. Summary

In summary, this section presented a survey of the research on ad hoc teamwork that is relevant to this article. A large amount of these works focus on simple domains and provide theoretical analyses. In addition, a substantial number of them assume that they know their teammates or share some communication or coordination protocols, but these works are still ad hoc teamwork because not all agents are designed by the same developers and the provided protocols are decentralized. However, these works do not consider the ad hoc teamwork problems investigated in this article, where the teammates may be completely unknown prior to the coordination. Finally, this article is the only work that we are aware of that learns about previous teammates and reuses this knowledge to quickly adapt to new teammates.

8. Conclusion

Given the growing numbers of agents in the world in the form of both robots and software agents, it is becoming a necessity for agents to cooperate to achieve their goals. However, cooperating with pre-designed teams is not enough; the agents will need to cooperate with a variety of teammates designed by other developers. Therefore, it is vital that the agents can reason about ad hoc teams, in which the agents adapt to their teammates on the fly.

This article investigates a limited version of the ad hoc teamwork problem, in which the ad hoc agent knows the environmental dynamics and has had previous interactions with other teammates and can exploit similarities between the teammates' behaviors. It is an ad hoc teamwork problem due to the fact that the ad hoc agent does not know the behavior of its teammates and its past teammates are not necessarily representative of its current teammates. PLASTIC reuses knowledge learned from past teammates and combines this knowledge with any advice provided by domain experts. This approach allows PLASTIC to quickly adapt to new teammates on the fly. We show that PLASTIC performs well on two disparate domains with a variety of teammates and differing amounts of knowledge about its teammates. While PLASTIC assumes that the current teammates' behaviors are similar to past teammates' behaviors, in our experiments, the behaviors were not explicitly designed to be similar. Instead, the teammates were created by a variety of independent developers. Nonetheless, PLASTIC was still able to discover similarities between these teammates' behaviors.

While this article covers many topics on research about ad hoc teams, it also raises many interesting questions. The teammates in this article are not explicitly learning about the ad hoc agent as it interacts with them. Therefore, one question is how to learn about these teammates while they are learning about the ad hoc agent. One approach to tackling this problem is for the ad hoc agent to maintain additional models of learning algorithms its teammates may be using. These models' learning algorithms would be updated with new observations in addition to updating the probabilities of the models relative to each other. Another approach is to consider the problem in the recursive modeling setting [20], whereby the ad hoc agent would reason about how its teammates are reasoning about it.

Additionally, PLASTIC may perform arbitrarily poorly when it encounters new teammates. PLASTIC relies on what it has learned about previous teammates, and if the new teammates' behaviors are arbitrarily far from these previous teammates, PLASTIC's performance may be arbitrarily bad. Falling back to learning from scratch is promising, but too slow for the time periods considered in this article. Instead, falling back to using a behavior with known bounds on the team's performance, like a safety strategy, is promising. However, calculating a behavior with bounded performance on ad hoc teamwork problems remains an open question.

A further interesting topic is to evaluate how well agents on ad hoc teams can learn about their environment as they also learn about their teammates. To simultaneously learn about unknown tasks and unknown teammates, an ad hoc team agent will need to balance the trade-off between exploiting its current knowledge, exploring the dynamics of the task, and exploring the behavior of its teammates. A possible approach to quickly learn in this scenario is for the ad hoc agent to also consider a set of previous environmental models to select from and update these models over time.

We look forward to future research along all these directions and believe that this article takes a significant step towards making such novel research on ad hoc teamwork possible.

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 (CNS-1330072, CNS-1305287), ONR (21C184-01), and AFOSR (FA8750-14-1-0070, FA9550-14-1-0087). This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1488/14), and by ERC Grant #267523. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[1] Agmon, N., Barrett, S., Stone, P., May 2014. Modeling uncertainty in leading ad hoc teams. In: Proceedings of the Thirteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[2] Agmon, N., Stone, P., June 2012. Leading ad hoc agents in joint action settings with multiple teammates. In: Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[3] Akiyama, H., 2010. Agent2d base code release. Http://sourceforge.jp/projects/rctools.

[4] Albrecht, S., Ramamoorthy, S., February 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. Tech. rep., School of Informatics, The University of Edinburgh, United Kingdom.

[5] Albrecht, S., Ramamoorthy, S., May 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems (extended abstract). In: Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS). St. Paul, Minnesota, USA.

[6] Albrecht, S., Ramamoorthy, S., July 2014. On convergence and optimality of best-response learning with policy types in multiagent systems. In: Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI). Quebec City, Canada.

[7] Albus, J. S., 1971. A theory of cerebellar function. Mathematical Biosciences 10 (12), 25 – 61.

[8] Albus, J. S., 1975. A new approach to manipulator control cerebellar model articulation control (CMAC). Transactions on ASME, J. of Dynamic Systems, Measurement, and Control 97 (9), 220–227.

[9] Almeida, F., Abreu, P. H., Lau, N., Reis, L., 2013. An automatic approach to extract goal plans from soccer simulated matches. Soft Computing 17 (5), 835–848.

[10] Bard, N., Johanson, M., Burch, N., Bowling, M., 2013. Online implicit agent modelling. In: Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 255–262.

[11] Barrett, S., Stone, P., June 2012. An analysis framework for ad hoc teamwork tasks. In: Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[12] Barrett, S., Stone, P., January 2015. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In: Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI).

[13] Barrett, S., Stone, P., Kraus, S., May 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In: Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[14] Barrett, S., Stone, P., Kraus, S., Rosenfeld, A., July 2013. Teamwork with limited knowledge of teammates. In: Proceedings of the Twenty-Seventh Conference on Artificial Intelligence (AAAI).

[15] Benda, M., Jagannathan, V., Dodhiawala, R., July 1986. On optimal cooperation of knowledge sources - An empirical investigation. Tech. Rep. BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services.

[16] Biswas, J., Mendoza, J. P., Zhu, D., Choi, B., Klee, S., Veloso, M., January 2014. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team. Proceedings of the Thirteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[17] Blum, A., Mansour, Y., 2007. Algorithmic Game Theory. Cambridge University Press, Ch. Learning, regret minimization, and equilibria.

[18] Bowling, M., McCracken, P., 2005. Coordination and adaptation in impromptu teams. In: Proceedings of the Twentieth Conference on Artificial Intelligence (AAAI). pp. 53–58.

[19] Brafman, R. I., Tennenholtz, M., 1996. On partially controlled multi-agent systems. Journal of Artificial Intelligence Research (JAIR) 4, 477–507.

[20] Carmel, D., Markovitch, S., 1996. Incorporating opponent models into adversary search. In: Proc. of AAAI. pp. 120–125.

[21] Chakraborty, D., Stone, P., June 2010. Convergence, targeted optimality and safety in multiagent learning. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML).

[22] Chakraborty, D., Stone, P., May 2013. Cooperating with a markovian ad hoc teammate. In: Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

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

[24] Dai, W., Yang, Q., Xue, G.-R., Yu, Y., 2007. Boosting for transfer learning. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML). pp. 193–200.

[25] Decker, K. S., Lesser, V. R., June 1995. Designing a family of coordination algorithms. In: International Conference on Multi-Agent Systems (ICMAS). pp. 73–80.

[26] Deisenroth, M. P., Neumann, G., Peters, J., 2013. A survey on policy search for robotics. Foundations and Trends in Robotics 2 (1-2), 1–142.

[27] Doshi, P., Zeng, Y., 2009. Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[28] Ernst, D., Geurts, P., Wehenkel, L., 2005. Tree-based batch mode reinforcement learning. In: Journal of Machine Learning Research (JMLR). pp. 503–556.

[29] Fang, M., Guo, Y., Zhang, X., Li, X., 2015. Multi-source transfer learning based on label shared subspace. Pattern Recognition Letters 51 (0), 101 – 106.

[30] Ge, L., Gao, J., Zhang, A., 2013. OMS-TL: A framework of online multiple source transfer learning. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management. CIKM '13. ACM, New York, NY, USA, pp. 2423–2428.

[31] Gelly, S., Wang, Y., December 2006. Exploration exploitation in Go: UCT for Monte-Carlo Go. In: Advances in Neural Information Processing Systems 19 (NIPS).

[32] Genter, K., Agmon, N., Stone, P., May 2013. Ad hoc teamwork for leading a flock. In: Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[33] Genter, K., Stone, P., September 2014. Influencing a flock via ad hoc teamwork. In: Proceedings of the Ninth International Conference on Swarm Intelligence (ANTS).

[34] Gmytrasiewicz, P. J., Doshi, P., Jul. 2005. A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research (JAIR) 24 (1), 49–79.

[35] Gmytrasiewicz, P. J., Durfee, E. H., Wehe, D. K., 1991. A decision-theoretic approach to coordinating multi-agent interactions. In: IJCAI. Vol. 91. pp. 63–68.

[36] Grosz, B., Kraus, S., 1996. Collaborative plans for complex group actions. Artificial Intelligence (AIJ) 86, 269–368.

[37] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H., November 2009. The WEKA data mining software: an update. SIGKDD Explorations 11, 10–18.

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

[39] Hausknecht, M., Mupparaju, P., Subramanian, S., Kalyanakrishnan, S., Stone, P., May 2016. Half field offense: An environment for multiagent learning and ad hoc teamwork. In: AAMAS Adaptive Learning Agents (ALA) Workshop. Singapore.

[40] Hester, T., Stone, P., 2013. TEXPLORE: real-time sample-efficient reinforcement learning for robots. Machine Learning (MLJ) 90 (3), 385–429.

[41] Hoang, T. N., Low, K. H., 2013. Interactive pomdp lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. In: The 23th International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press, pp. 2298–2305.

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

[43] Huang, Y.-W., Sasaki, Y., Harakawa, Y., Fukushima, E., Hirose, S., sept. 2011. Operation of underwater rescue robot anchor diver III during the 2011 Tohoku earthquake and tsunami. In: OCEANS 2011. pp. 1 –6.

[44] Isaacs, R., 1965. Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization. Dover Publications.

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

[46] Jones, E., Browning, B., Dias, M. B., Argall, B., Veloso, M. M., Stentz, A. T., May 2006. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). pp. 570 – 575.

[47] Jung, T., Polani, D., Stone, P., 2010. Empowerment for continuous agent-environment systems. Tech. Rep. AI-10-03, The University of Texas at Austin Computer Science Department.

[48] Kalyanakrishnan, S., Liu, Y., Stone, P., 2007. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In: RoboCup-2006: Robot Soccer World Cup X. Vol. 4434 of Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, pp. 72–85.

[49] Kalyanakrishnan, S., Stone, P., July 2011. Characterizing reinforcement learning methods through parameterized learning problems. Machine Learning (MLJ) 84 (1–2), 205–247.

[50] Kamishima, T., Hamasaki, M., Akaho, S., Dec. 2009. TrBagg: A simple transfer learning method and its application to personalization in collaborative tagging. In: Ninth IEEE International Conference on Data Mining (ICDM). pp. 219 –228.

[51] Kocsis, L., Szepesvari, C., 2006. Bandit based Monte-Carlo planning. In: Prooceedings of the Seventeenth European Conference on Machine Learning (ECML).

[52] Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., Tambe, M., May 2011. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. Journal of Artificial Intelligence Research (JAIR) 41 (2), 297–327.

[53] Lauer, M., Riedmiller, M., 2000. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML). Morgan Kaufmann, pp. 535–542.

[54] Liemhetcharat, S., Veloso, M., 2014. Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents. Artificial Intelligence (AIJ) 208 (0), 41 – 65.

[55] Murphy, R., Dreger, K., Newsome, S., Rodocker, J., Steimle, E., Kimura, T., Makabe, K., Matsuno, F., Tadokoro, S., Kon, K., November 2011. Use of remotely operated marine vehicles at Minamisanriku and Rikuzentakata Japan for disaster recovery. In: Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on. pp. 19 –25.

[56] Nagatani, K., Kiribayashi, S., Okada, Y., Tadokoro, S., Nishimura, T., Yoshida, T., Koyanagi, E., Hada, Y., November 2011. Redesign of rescue mobile robot Quince. In: Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on. pp. 13 –18.

[57] Pardoe, D., Stone, P., June 2010. Boosting for regression transfer. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML).

[58] Richardson, D., May 2011. Robots to the rescue? Engineering Technology 6 (4), 52 –54.

[59] Silver, D., Sutton, R. S., Müller, M., 2008. Sample-based learning and search with permanent and transient memories. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML).

[60] Silver, D., Veness, J., 2010. Monte-Carlo planning in large POMDPs. In: Advances in Neural Information Processing Systems 23 (NIPS).

[61] Sonu, E., Doshi, P., 2012. Generalized and bounded policy iteration for finitely-nested interactive POMDPs: Scaling up. In: Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 1039–1048.

[62] Stone, P., 2000. Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press.

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

[64] Stone, P., Kaminka, G. A., Rosenschein, J. S., November 2010. Leading a best-response teammate in an ad hoc team. In: AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC).

[65] Stone, P., Kraus, S., May 2010. To teach or not to teach? Decision making under uncertainty in ad hoc teams. In: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

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

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

[68] Tambe, M., 1997. Towards flexible teamwork. Journal of Artificial Intelligence Research (JAIR) 7, 81–124.

[69] Taylor, M. E., Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research (JMLR) 10 (1), 1633–1685.

[70] Undeger, C., Polat, F., 2010. Multi-agent real-time pursuit. Autonomous Agents and Multi-Agent Systems (JAAMAS) 21 (1), 69–107.

[71] Watkins, C. J. C. H., May 1989. Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK.

[72] Wu, F., Zilberstein, S., Chen, X., 2011. Online planning for ad hoc autonomous agent teams. In: The 22th International Joint Conference on Artificial Intelligence (IJCAI).

[73] Xuan, P., Lesser, V., Zilberstein, S., 2001. Communication decisions in multiagent cooperation: model and experiments. In: Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS).

[74] Yao, Y., Doretto, G., June 2010. Boosting for transfer learning with multiple sources. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR).

[75] Zeng, Y., Chen, Y., Doshi, P., 2011. Approximating model equivalence in interactive dynamic influence diagrams using top k policy paths. Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on 2, 208–211.

[76] Zeng, Y., Doshi, P., Jan. 2012. Exploiting model equivalences for solving interactive dynamic influence diagrams. Journal of Artificial Intelligence Research (JAIR) 43 (1), 211–255.

[77] Zhuang, F., Cheng, X., Pan, S., Yu, W., He, Q., Shi, Z., 2014. Transfer learning with multiple sources via consensus regularized autoencoders. In: Calders, T., Esposito, F., Hllermeier, E., Meo, R. (Eds.), Machine Learning and Knowledge Discovery in Databases. Vol. 8726 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 417–431.

[78] Zinkevich, M., Johanson, M., Bowling, M., Piccione, C., 2008. Regret minimization in games with incomplete information. In: Advances in Neural Information Processing Systems 20 (NIPS). pp. 905–912.

Appendix A. Markov Decision Process Algorithms

This appendix provides descriptions of the algorithms used to solve MDPs in the results presented in Sections 5 and 6. Note that other algorithms for solving MDPs can be used in PLASTIC, and we recommend selecting algorithms suited for the domain considered.

Appendix A.1. Value Iteration

One way to calculate the optimal policy is by using Value Iteration (VI) [67]. VI requires a complete model of the environment, specifically the full transition and reward functions. Given these models, VI can calculate the optimal value function $Q^*(s, a)$ and therefore the optimal policy π^*. Value iteration relies on dynamic programming to solve for 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 given in Equation 3.1. These updates are repeated iteratively until convergence, and the final calculated state-action values are guaranteed to be optimal.

While VI provably converges to the optimal policy, this convergence may take a substantial amount of time. VI has difficulties in scaling to large domains as it requires visiting each state-action over many iterations. In ad hoc teamwork scenarios, this problem is especially costly as the number of agents greatly increases the state space. In many problems, the team's state space becomes exponential in the size of the domain, with a power proportional to the number of agents, (# positions)$^{(\text{\# agents})}$. Given the symmetries of a specific problem, it is sometimes possible to reduce the number of possible states, but the scaling is still poor. For example, in initial tests into ad hoc teamwork in the pursuit domain (described in Section 2.4), VI on a 5x5 world took approximately 12 hours on the University of Texas Mastodon computing cluster. In a 5x5 world, there are $25^5 \approx 1e7$ states to consider ignoring symmetries, given that there are 5 agents moving around the 25 world positions. Scaling up to a larger problem of a 20x20 world, there are $400^5 \approx 1e13$ states of the entire team. Thus, there are more than a million times more states than the 5x5 world, leading to this problem to be computationally infeasible. Due to the exponential blowup of the size of the state space, many ad hoc teamwork problems are not suitable for VI, even if the teammates' behaviors are fully known and the problem can be described as an MDP.

Appendix A.2. Monte Carlo Tree Search

Value Iteration is one approach for solving MDPs, and it would allow an ad hoc agent to optimally cooperate with its teammates if were it to have a complete model of its teammates and the environment. However, VI is often infeasible to run in a reasonable time and requires a complex model. Rather than calculating the exact optimal value of every state-action, it is much more computationally tractable to instead learn an approximately optimal value for relevant state-actions. 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, the MCTS algorithm called Upper Confidence bounds for Trees (UCT) [51] is used as a starting point for creating the primary planning algorithm used in this article.

MCTS does not require a complete model of the environment. Rather than knowing the full probability distribution of next states and rewards resulting from the transition and reward functions, MCTS only needs a model that allows sampling these next states and rewards. Furthermore, rather than treating all of the state-actions as equally likely, UCT focuses on only calculating 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. When selecting actions, UCT greedily chooses the action with the highest upper confidence bound. UCT has been shown to be effective on domains with a high branching factor, such as Go [31] and large POMDPs [60]. As such, we reasoned it should be able to handle the branching factor caused by the number of agents.

In this article, UCT is modified to use eligibility traces and remove the depth index to help speed up learning. The pseudocode of the algorithm can be seen in Algorithm 5, with s being the current state. Similar modifications were made by Silver et al., with good success in Go [59]. In addition, work by Hester and Stone [40] show good results in a number of other reinforcement learning domains.

Appendix A.3. Fitted Q Iteration

While UCT is effective for quickly computing an approximately optimal policy in an MDP, it does require a model of the MDP that permits sampling from the transition and reward functions. This model can either be given or learned given enough data. VI requires a stronger model; a model that gives the full probability distribution of next states and rewards for the transition and reward functions. However, other approaches attempt to directly learn the values of state-actions without a model of the transition function, and these approaches are called *model-free*. Model-free approaches do not require building a model of the domain which can be more tractable in hard to model domains. In addition, model-free algorithms are often computationally simpler. In complex domains, it may be difficult for ad hoc agents to compute the model of their environment and teammates, so it may be useful for the ad hoc agent to employ a model-free learning method to find a good policy for cooperating with its teammates.

In this work, our agent uses the Fitted Q Iteration (FQI) algorithm introduced by Ernst et al. [28]. Similar to Value Iteration (VI), FQI iteratively backs up rewards to improve its estimates of the values of states. Rather than looking at every state and every possible outcome from each state, FQI uses samples of these states and outcomes to approximate the values of state-action pairs. This approximation allows FQI to find solutions for complex, continuous domains. Alternative policy learning algorithms can be used, such as Q-learning [71] or policy search [26].

To collect samples of the domain, the agent first performs a number of exploratory actions. From each action, the agent stores the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state. An advantage of the

Algorithm 5 The modified version of UCT used in this article

```
1:  function UCTSelect:
        inputs:
          $s$                                         ▷ the current state
        outputs:
          $a$                                         ▷ action selected by UCT
        params:
          $\gamma$                                    ▷ discount factor, parameter of the MDP
          NumPlayouts                                 ▷ number of Monte Carlo playouts to perform
          $c$                                         ▷ weight given to the confidence bound
          $\lambda$                                   ▷ eligibility trace parameter - affects amount of backup
          simulateAction$(s, a)$                      ▷ an environment model that samples next states

2:      for $i = 1$ to NumPlayouts do
3:          Search$(s)$
4:      return $a = \text{argmax}_a\ Q(s, a)$

5:  function Search$(s)$:
6:      $a =$ bestAction$(s)$
7:      while $s$ is not terminal do
8:          $(s', r) =$ simulateAction$(s, a)$
9:          $a' =$ bestAction$(s')$
10:         $e(s, a) = 1$
    ▷ Update the Q-values
11:         $\delta = r + \gamma Q(s', a') - Q(s, a)$
12:         for all $s^*, a^*$ do
13:             $Q(s^*, a^*) = Q(s^*, a^*) + e(s^*, a^*) * \frac{\delta}{\text{visits}(s^*, a^*)}$
14:             $e(s^*, a^*) = \lambda e(s^*, a^*)$
15:         $s = s'; a = a';$

16: function bestAction$(s)$:
17:     return $\text{argmax}_a\ Q(s, a) + c\sqrt{\frac{\ln \text{visits}(s)}{\text{visits}(s, a)}}$
```

FQI algorithm is that this data can be collected in parallel from a number of tests. At each iteration, the agent updates the following equation for each tuple

$$Q(s,a) = r + \gamma * \max_{a'} Q(s',a')$$

where $Q(s,a)$ is initialized to 0. Q is an estimate of the optimal value function, Q^*, and this estimate is iteratively improved by looping over the stored samples. To handle continuous state spaces, Q is not stored exactly in a table; instead, its value is estimated using function approximation. In this article, the continuous state features are converted into a set of binary features using CMAC tile-coding [7, 8], and the estimate of $Q(s,a)$ is given by

$$\hat{Q}(s,a) = \sum_i w_i f_i$$

where f_i is the i^{th} binary feature and w_i is the weight given to the feature with updates split uniformly between the active features. This approach uses a set of overlapping tilings to cover the space with binary activations [67]. The advantages of tile coding include simple computation, binary output, and good control over the generalization of the model.