To appear in *AAMAS Autonomous Robots and Multirobot Systems Workshop* (ARMS), Paris, France, May 2014.

Cooperating with Unknown Teammates in Robot Soccer

Samuel Barrett and Peter Stone
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
{sbarrett,pstone}@cs.utexas.edu

No Institute Given

Abstract. Many scenarios require that robots work together as a team in order to effectively accomplish their tasks. However, pre-coordinating these teams may not always be possible given the growing number of companies and research labs creating these robots. Therefore, it is desirable for robots to be able to reason about *ad hoc teamwork* and adapt to new teammates on the fly. This paper adopts an approach of learning models of past teammates and reusing these models to adapt to the new teammates. Then, these models are used to plan actions that lead the team towards their shared goal. This approach is applied to the complex domain of robot soccer in the form of half field offense in the RoboCup simulated 2D league. This paper represents a preliminary investigation into this domain and presents a promising approach for tackling this problem.

1 Introduction

As the presence of robots grows, so does their need to cooperate with one another. Usually, multiagent research focuses on the case where all of the robots have been given shared coordination protocols before they encounter each other [1–3]. However, given the growing number of research laboratories and companies creating new robots, not all of these robots may share the same coordination protocols. Therefore, it is desirable for these robots to be capable of learning and adapting to new teammates. This area of research is called *ad hoc teamwork* and focuses on the scenario in which the developers only design a single agent or small subset of agents that need to adapt to a variety of teammates [4].

Previous research into ad hoc teamwork has established a number of theoretical and empirical methods for handling unknown teammates, but these analyses largely focus on simple scenarios that may not be representative of the real problems that robots may encounter in ad hoc teams. This paper addresses this gap by investigating ad hoc teamwork in the RoboCup 2D simulation domain, a complex, multiagent domain in which teams of agents coordinate their actions in order to compete against other intelligent teams. This domain requires effective cooperation between the agents while acting in a world with continuous states and continuous actions, all while facing complex opponents. The main contribution of this paper is the introduction of an algorithm for planning effective actions for cooperating with a variety of teammates in this domain as well as the empirical evaluation of this algorithm. This paper serves as a preliminary exploration of ad hoc teamwork in this complex domain and discusses a promising approach for creating intelligent ad hoc team agents for this problem.

To quickly adapt to new teammates, it is imperative to use knowledge learned about previous teammates. This paper adopts the method of learning models of how previous teammates act in the world and then uses these models to plan to cooperate with new teammates. These models are represented as mappings from current world states to the teammates' actions. Specifically, the problem of learning these models is treated as a supervised learning problem where the inputs are features derived from the current world state and labels are a teammate's actions.

The remainder of this paper is organized as follows. Section 2 situates this research in the literature, and then Section 3 describes the problem in more depth. Section 4 describes the methods used to tackle this problem, and Section 5 concludes.

2 Related Work

Multiagent teamwork is a well studied area of research. Previous research into multiagent teams has largely focused on creating shared protocols for coordination and communication which will enable the team to cooperate effectively. For example, the STEAM [2] framework has agents build a partial hierarchy of joint actions. In this framework, agents monitor the progress of their plans and adapt their plans as conditions change while selectively using communication in order to stay coordinated. Alternatively, the Generalized Partial Global Planning (GPGP) [5] framework considers a library of coordination mechanisms from which to choose. Decker and Lesser argue that different coordination mechanisms are better suited to certain tasks and that a library of coordination mechanisms is likely to perform better than a single monolithic mechanism. In SharedPlans [6], agents communicate their intents and beliefs in order to reason about coordinating joint actions, while revising these intents as conditions change.

While pre-coordinated multiagent teams are well studied, there has been less research into teams in which this pre-coordination is not available. This work builds on the ideas presented by Barrett et al. [7], specifically learning models about past teammates and using these models to quickly adapt to new teammates. However, that work focused on a simpler domain in the form of a grid world, while this work investigates a complex, simulated robotics domain. One early exploration of ad hoc teamwork is that of Brafman and Tennenholtz [8] in which a more experienced agent must teach its novice teammate in order to accomplish a repeated joint task. Another line of research, specifically in the RoboCup domain, was performed by Bowling and McCracken [9]. In their scenario, an agent has a different playbook from that of its teammates and must learn to adapt. Their agent evaluates each play over a large number of games and predicts the roles that its teammates will adopt.

Other research into ad hoc teams includes Jones et al.'s [10] work on pickup teams cooperating in a domain involving treasure hunts. Work into deciding which teammates should be selected to create the best ad hoc team was performed by Liemhetcharat and Veloso [11]. Additional work includes using stage games and biased adaptive play to cooperate with new teammates [12]. However, the majority of these works focus on simple domains and provide the agent with a great amount of expert knowledge.

A very closely related line of research is that of opponent modeling, in which agents reason about opponents rather than teammates. This area of research largely focuses on bounding the worst case scenario and often restricts its focus onto repeated matrix games. One interesting approach is the AWESOME algorithm [13] which achieves convergence and rationality in repeated games. Valtazanos and Ramamoorthy explore modeling adversaries in the RoboCup domain in [14]. Another approach is to explicitly model and reason about other agents' beliefs such as the work on I-POMDPs [15], I-DIDs [16], and NIDs [17]. However, modeling other agents' beliefs greatly expands the planning space, and these approaches do not currently scale to larger problems.

3 Problem Description

In this paper, we consider the scenario in which an ad hoc agent must cooperate with a team of agents that it has never seen before. If the agent is given coordination or communication protocols beforehand, it can easily cooperate with its teammates. However, in scenarios where this prior shared knowledge is not available, the agent should still be able to cooperate with a variety of its teammates. Specifically, an agent should be able to leverage its knowledge about previous teammates in order to help it when it encounters new teammates.

3.1 Ad Hoc Teamwork

Evaluating an ad hoc team agent is difficult because the evaluation relies on how well the agents can cooperate with various teammates; we only create a single agent rather than an entire team. Therefore, to estimate the performance of agents in this setting, we adopt the evaluation framework proposed by Stone et al. [4]. In this work, Stone et al. formulate the performance of an ad hoc team agent as explicitly depending on the teammates and domains that it may encounter. Therefore, we describe the domain and teammates that the ad hoc team agents will encounter in the following sections.

3.2 RoboCup 2D Simulation Domain

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 for two 5 minute halves. The game lasts for 6,000 simulation steps, each lasting 100 ms. At each of these steps, these 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. 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.

3.3 Half-Field Offense

Rather than use full 10 minute 11 on 11 game, this work instead uses the quicker task of half field offense introduced by Shivaram et al. [18]. In Half Field Offense (HFO), 4 offensive agents attempt to score on 5 defensive ones, including a goalie, without letting the defense capture the ball. A view of this game is shown in Figure 1. 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.

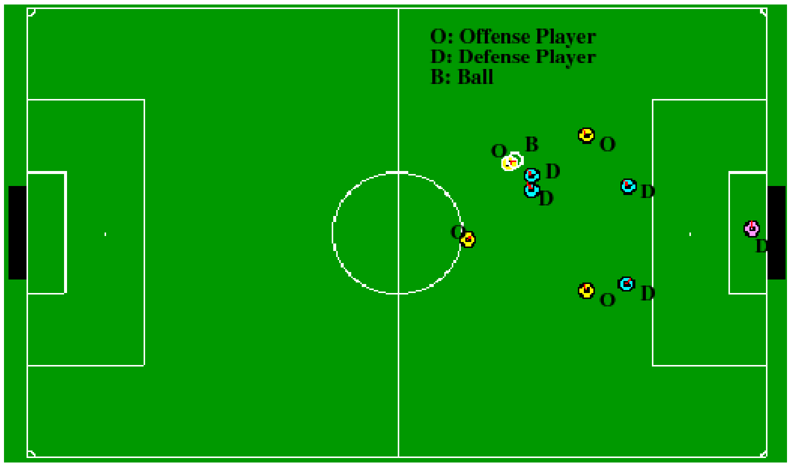

Fig. 1: A screenshot of half field offense in the 2D soccer simulation league.

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 soccer pitch. Offensive players start in 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.

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 affect the players used in the HFO, but empirical tests have shown that the teams still perform well and that our ad hoc team agent can still adapt to these teams.

3.4 Teammates

In ad hoc teamwork research, it is important to use *externally-created* teammates to evaluate the various ad hoc team agents. Externally-created teammates are created by developers other than the authors and represent real agents that are created for the domain when developers do not plan for ad hoc teamwork scenarios. They are useful because their development is not biased to make them more likely to cooperate with ad hoc team agents. As part of the 2D simulation league competition, teams are required to release binary versions of their agents following the competition. Specifically, we use the binary releases from the 2013 competition. These agents provide an excellent source of *externally-created* teammates with which to test the possible ad hoc team agents.

4 Methods

Intelligent behaviors for cooperating with teammates can be difficult to design, even if the teammates' behaviors are known. In the 2D simulation league, there is the added difficulty of the domain being noisy and requiring many low level actions to accomplish any given task. To address this challenge, we propose that the agent employ a probabilistic planner that uses a stochastic model of the domain and learns models of its teammates.

4.1 MDP Formulation

Before specifying how to approach this ad hoc teamwork problem, it is helpful to describe how to model the problem. Specifically, we model the problem as a Markov Decision Process (MDP), which is a standard formalism in reinforcement learning [19] for describing an agent interacting with its environment. 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 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 maximizes 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 be derived by choosing the action a that maximizes $Q^*(s,a)$ for every $s \in S$. We now describe how to model the HFO domain as an MDP.

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. Currently, the noiseless versions of these values are used, but the full noisy estimations provided by the simulator will be used in future work.

Actions In the 2D simulation league, agents act by selecting whether to dash, turn, or kick and specify values like the power and angle to kick at. Combining these actions to accomplish the desired results is a difficult problem. Therefore, this work builds on the code release by Helios [20]. This code release provides for a number of high level actions, such as passing, shooting, or moving to a specified point.

We use a set of 7 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 a short distance, maintaining control of the ball.
3. Long dribble – dribble the ball a longer distance, which will require chasing the ball down.
4. Cross – perform a crossing pass near the goal.
5. Pass_0 – perform a direct, leading, or through pass to teammate 0.
6. Pass_1 – perform a direct, leading, or through pass to teammate 1.
7. Pass_2 – perform a direct, leading, or through 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 Helios evaluation function. While using these high level actions restricts the possibilities that the agent can take, it also enables it to plan more quickly and prune out ineffective actions, allowing it to select more intelligent actions in the constrained time available.

Transition Function 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 effects of our actions are predicted using the Helios code base, with some added Gaussian noise to model the sensing and actuation noise of the simulator. We learn a model of the other agents' actions as described in Section 4.2.

Reward Function 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 completely outweigh these effects. Other values were tested with similar results.

4.2 Modeling Other Agents

Modeling teammates is more difficult; in this work, the agent learns models of past teammates. These models are restricted to the case in which the teammate does not possess the ball and take the form of predicting where the teammate will move. The agent learns these models by observing past teammates and storing which actions the teammates took in each world state for each action the agent can take. Then, it treats this problem as a supervised learning problem, where the actions are the labels and the features are derived from the world state.

Specifically, the features used for these predictions are:

- Duration – the number of steps to predict into the future
- Position – the agent's x and y position on the field
- Orientation – the direction that the agent is facing
- Ball position – the relative position of the ball
- Ball velocity – the ball's velocity
- Ad hoc position – the relative position of the ad hoc player
- Team reach time – the number of steps for a teammate to reach the ball
- Opponent reach time - the number of steps for an opponent to reach the ball

These features result in a total of 12 values since the velocities and positions are two dimensional. The reach times of the agents are estimated using the code in the Helios release [20]. There are three values predicted for each of these points: (1) the change in x position, (2) the change in y position, and (3) the change in orientation. All features and labels are then scaled to the range $[0, 1]$ using the observed range of values. Prediction can be done using any number regression algorithms. After evaluating several methods, linear regression was chosen as it performed well empirically and its speed of computation allows for greater numbers of Monte Carlo simulations to be run in the allotted time (explained in Section 4.3. More complex methods can be used for more accurate predictions, but the accuracy is still limited due to the complexity of the agents' decision functions. Therefore, running more rollouts proved to be more effective than predicting more accurately.

These predictions are learned for each action that the ad hoc agent may take. Multiple models are used because the ad hoc agent's actions are high level and take several steps of the simulator to complete. Therefore, the teammates can perceive and react to the agent's actions as they execute. For example, the teammates can perceive if the ad hoc agent is dribbling the ball versus passing it, which will cause them to react differently.

Different models can be learned of different teammates that have been encountered in the past. When the ad hoc agent encounters a new team, it can attempt to see which model it has learned applies to the new teammates. Specifically, the agent can maintain a belief distribution over the past models and update these beliefs based on the probability that the model would have predicted the observed actions. This approach has been shown to be effective in previous work [7] even when the models are inaccurate and do not match the current teammates' behaviors.

These predictions are only of how the teammates move when they do not have the ball. When a teammate has the ball, the agent plans as if that agent acted similarly to itself. In other words, it expects its teammates to take the actions that its planning expects to be effective when they hold the ball. While this assumption may be wrong, initial tests indicate that this approach is effective. Learning models of how different teammates act with the ball is left to future work.

While all of the description above discusses the learning of models of teammates, this approach is also used to learn models of the opponents. The main difference is that ad hoc agent does not need to select which model of the opponents to use as there is only a single type of opponents in this setting. Simultaneously handling multiple types of opponents is an interesting problem, but is left to future work.

4.3 Planning

The previous section describes how to learn models of teammates, but not how to use these models to determine which actions to take. Planning allows us to convert the knowledge represented by these models into intelligent behaviors. However, given the large continuous state of the half field offense domain, planning is a difficult problem. To address this problem, we adopt a modified version of the Upper Confidence bounds for Trees (UCT) algorithm [21]. UCT is a Monte Carlo Tree Search (MCTS) algorithm which has been shown to handle domains with large state spaces with high branching factors, such as Go [22] and large POMDPs [23]. UCT enables us to estimate the values of taking different actions from states that are likely to be encountered. These estimations are built using a number of Monte Carlo simulations that sample outcomes of taking various actions.

Note that all of these models are single step (at the high level view); they only predict the immediate actions of the other agents. However, it is possible to combine these predictions into long term predictions by using the predictions of each model as input into the following predictions. This approach does accumulate errors from the previous predictions, but repeating these predictions many times should ideally balance out this noise.

In UCT, the agent samples a number of simulations from the current game state and stores information about the values of states and actions in a tree. In addition to these values, the agent also stores the number of times it has tried each action from each state. In continuous domains, it is helpful to store these values as a discretized function in order to combine states that are very close to each other, though not exactly the same. This discretization is performed by binning the position, velocities, and orientations of the agents into bins of size 0.1m and 0.1 radians. However, the simulations are performed in the original, continuous state; the states are only discretized for the value and visit functions. During the simulations, at states it has previously visited, the agent selects actions using upper confidence bounds, which allows for intelligently trading off between exploration and exploitation.

In addition, we further modify how UCT updates the values of state-actions. By default, UCT only uses the reward accumulated during the Monte Carlo rollout. In contrast, we complement this information with the current estimates of the values of state-actions using the eligibility trace updates used in Q(λ) learning. Let $Q(s, a)$ be the estimate of the value of taking action a from state s, n_{sa} be the number of visits of the state-action pair, s' be the resulting state, and $0 \leq \gamma \leq 1$ be the discount factor of future rewards r_i. Then the new value is given by:

$$\delta_{t+1} = \lambda \left(\sum_i \gamma^i r_i \right) + (1 - \lambda) \max_{a'} Q_t(s', a') - Q_t(s, a)$$

$$Q_{t+1}(s, a) = Q_t(s, a) + \frac{1}{n_{sa} + 1} \delta_{t+1}$$

where λ balances Monte Carlo versus Temporal Difference (TD) updates. $\max_{a'} Q_t(s', a')$ is the maximum value over the experienced actions from the next state including actions experienced in the current rollout. Intuitively, this approach allows UCT to converge to the true value function more quickly, resulting in selecting better actions.

Furthermore, we drop the depth of states (number of actions required to reach the state) usually used in the UCT tree. This change combines several states and may turn the tree into a graph. However, combining more states allows the planner to better estimate the values of state-action pairs by having more visits to the state-action. Finally, UCT estimates the upper confidence bound using $Q(s,a) + C_p\sqrt{(\ln n_s)/n_{sa}}$, where n_s is the number of visits of the state and $C_p = 2^{1/2}$ when the MDP rewards are scaled between 0 and 1. Empirically, it is useful to tune the value of C_p in order to better balance exploration versus exploitation given that the number of rollouts is limited due to computational constraints.

5 Conclusion

The majority of past research on ad hoc teamwork has focused on domains such as grid worlds or matrix games and in which only a few agents interact. However, the dream of applying ad hoc teamwork techniques to real world problems requires scaling these methods up to more complex and noisy domains that involve many agents interacting. This work presents a step towards this goal by investigating ad hoc teamwork in the area of simulated robot soccer. This research is a preliminary report focusing on how to approach this complex problem. This paper presents an approach for learning models that predict teammates' actions as an effective way of representing knowledge learned from past teammates. Then, we show how this knowledge can be used to plan intelligent behaviors using a sample-based planner to select high level actions. Additionally, we show how to modify this planner to handle continuous states effectively.

This paper represents a move towards scaling up reasoning about ad hoc teamwork to a complex domain involving many agents. One area for future work is applying this approach to real robots; for example, in future drop-in player challenges run at the RoboCup competition, such as those held in 2013 in the 2D simulation league, the 3D simulation league, and the Standard Platform League (SPL). Another interesting avenue for future research is reasoning about agents that are learning about the ad hoc agent over time. In the long term, an interesting application of ad hoc teamwork is to allow agents to cooperate with either other agents or humans by considering them as additional agents in the domain.

References

1. Grosz, B.J., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence **86**(2) (1996) 269–357
2. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research **7** (1997) 83–124
3. Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., Garvey, A.: The TAEMS White Paper (January 1999)
4. Stone, P., Kaminka, G.A., Kraus, S., Rosenschein, J.S.: Ad hoc autonomous agent teams: Collaboration without pre-coordination. In: AAAI '10. (July 2010)
5. Decker, K.S., Lesser, V.R.: Designing a family of coordination algorithms. In: ICMAS '95. (June 1995) 73–80

6. Grosz, B., Kraus, S.: Collaborative plans for complex group actions. Artificial Intelligence **86** (1996) 269–368
7. Barrett, S., Stone, P., Kraus, S., Rosenfeld, A.: Teamwork with limited knowledge of teammates. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. (July 2013)
8. Brafman, R.I., Tennenholtz, M.: On partially controlled multi-agent systems. Journal of Artificial Intelligence Research **4** (1996) 477–507
9. Bowling, M., McCracken, P.: Coordination and adaptation in impromptu teams. In: AAAI. (2005) 53–58
10. Jones, E., Browning, B., Dias, M.B., Argall, B., Veloso, M.M., Stentz, A.T.: Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In: ICRA. (May 2006) 570 – 575
11. Liemhetcharat, S., Veloso, M.: Modeling and learning synergy for team formation with heterogeneous agents. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. AAMAS '12, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2012) 365–374
12. Wu, F., Zilberstein, S., Chen, X.: Online planning for ad hoc autonomous agent teams. In: IJCAI. (2011)
13. Conitzer, V., Sandholm, T.: AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. Machine Learning **67** (May 2007)
14. Valtazanos, A., Ramamoorthy, S.: Bayesian interaction shaping: Learning to influence strategic interactions in mixed robotic domains. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems. AAMAS '13, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2013) 63–70
15. Gmytrasiewicz, P.J., Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research **24**(1) (July 2005) 49–79
16. Doshi, P., Zeng, Y.: Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In: AAMAS '09. (2009)
17. Gal, Y., Pfeffer, A.: Network of influence diagrams: Reasoning about agents' beliefs and decision-making processes. Journal of Artificial Intelligence Research **33** (2008) 109–147
18. Kalyanakrishnan, S., Liu, Y., Stone, P.: Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In: RoboCup-2006: Robot Soccer World Cup X. Volume 4434 of Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin (2007) 72–85
19. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA (1998)
20. Akiyama, H.: Agent2d base code release (2010) http://sourceforge.jp/projects/rctools.
21. Kocsis, L., Szepesvari, C.: Bandit based Monte-Carlo planning. In: ECML '06. (2006)
22. Gelly, S., Wang, Y.: Exploration exploitation in Go: UCT for Monte-Carlo Go. In: NIPS '06. (December 2006)
23. Silver, D., Veness, J.: Monte-Carlo planning in large POMDPs. In: NIPS '10. (2010)