Stereotypes in Coordination Domains

Carrie Rebhuhn
Oregon State University
rebhuhnc@onid.orst.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

In coordination tasks with heterogeneous agents it can be critical to have a representation of another agent's abilities. However in large systems it is impossible to keep a complete model of other agents. We show that in coordination tasks, using a generalized agent model, or *stereotype*, can increase the performance of an agent. We demonstrate this in an n-agent pursuit domain.

1. INTRODUCTION

Real systems are heterogeneous. Many multiagent problems treat other agents as background noise, insensitive to the differences between them. In some cases this approach makes sense, as the actions of agents are decoupled enough that they do not need to consider the action of its collaborators. In domains with higher coupling and coordination requirements, knowledge of an agent's action preferences or capability differences can be key to finding optimal actions within the system.

Agent modeling techniques accommodate these differences. Early work focused on modeling the actions of other agents in order to predict and exploit their next move. However this approach comes at a considerable cost when encountering more than one other agent. If a multiagent system is large, it can become costly for each agent to create a separate model of another agent. Modeling each agent individually can also hinder performance if an agent has brief interactions with another agent in the system, and cannot take advantage of the model.

An approach which has had more success on larger systems is the approach of *generalized* agent modeling. This approach dictates a fundamental set of agent models, and each agent in the system is fitted to one of these model. This becomes much more scalable because the number of models in the system does not grow with the number of agents. This also allows model information to be used on any agent, whether encountered previously or not, with a given confidence about an agent's type identification. However this approach requires prior knowledge about the model types existing in the system. This can lead to issues in identifying coordinating actions if there are unmodeled behaviors, and leaves no room to accommodate adaptation within the agent type.

We take the approach that an agent can learn using an abstract type identification. Instead of incorporating an explicit model defined by a type of agent and then reasoning about each agents in the environment, we simply identify each agent with a type, and then incorporate this information into the state space of the rover using a demographics approach (percent each agent in surroundings). We demonstrate the efficacy of this approach in two domains; the first a simple rover domain, and the second a more complex air traffic conflict-avoidance domain. This demonstrates the efficacy of using minimal but nonzero type information in both simple and more complex coordination domains.

Our approach has the advantages of:

- **Simplicity**: We show that by including even an abstract identification of type in the state space, we can leverage information about another agent. This allows for use of any classifier. We demonstrate that classification techniques can be leveraged in simple learning without concept reasoning about the state representation.

- **Scalability**: Unlike traditional game theoretic representations, the optimal joint action does not need to be calculated. Our approach uses a very abstract identification of types (which is represented by the ratio of types in the different quadrants). This input is able to scale with the number of agents without increasing the complexity of the state space.

- **Graceful Degradation**: In many approaches using agent modeling, if a type is misidentified then the agent will reason incorrectly about its most desirable coordination action and it may make a severe mistake. Because our approach uses an adaptive model (a neural network) the potential for incorrectly identifying is implicitly considered, therefore minimizing the cumulative effect of mistaken identifications.

Because of these contributions we can extend the use of agent modeling in large domains where it has previously been computationally intractable.

The paper is structured as follows: Section 2 outlines related work on agent modeling and coordination domains. Section 3 will describe the method we use for our experiments. Section 4 will discuss the findings of our experiments, and Section 5 will discuss the implications of our results and directions for future work.

Appears in: *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), Lomuscio, Scerri, Bazzan, Huhns (eds.), May, 5–9, 2014, Paris, France.*

2. RELATED WORK

Stereotyping has varying definitions, but we define it as grouping several agents under a single model definition. By identifying another agent's policy, an agent can make a more informed decision. This has not been well researched in domains with large numbers of agents, and is the key focus of this work.

We first provide an overview of agent modeling in Section 2.1. We then discuss stereotypes in Section 2.2. Finally, we give background on neural evolution and the pursuit domain in Section 2.3.

2.1 Agent Modeling

It is intuitive that knowing information about your opponent could give some advantage against them, but the real question is how to compactly represent this with enough detail to gain maximum advantage. Early work in poker produced two algorithms using agent modeling with his *Loki* poker-playing program, Generic Opponent Modeling (GOM) and Specific Opponent Modeling (SOM) which keep track of probabilities that each player will play a hand and continuously update these. GOM did not diffrentiate between the poker players, while SOM created a model for each player. GOM and SOM proved able to outperform algorithms without modeling [1].

Chakraborty et al. models memory-bounded agents using high-level features derived from a fixed number of past actions called Targeted Opponent Modeling for MEmory-Bounded Agents (TOMMBA) [5]. Ponsen et al. propose a Bayes-relational opponent model that starts with a set of priors also, but then use this to learn a relational regression tree to adapt these to specific opponents [9]. Williams et al. determine when to make concessions by using a Gaussian process to estimate the future play of an opponent [12]. Early work by Carmel et al. develops an extension to L^* by approaching modeling agents as a dimensional reduction of a deterministic finite automaton (DFA), and then uses this model to predict and exploit its next action [4].

Training examples are used from previous playes to train the classifier. The classifier then takes state information as a feature vector and returns the predicted output action. Ekmekci and Sirin use three different machine learning techniques; neural networks, support vector machines, and K nearest neighbors, in order to classify opponent move sin poker [6]. Laviers et al. used an SVM classifier to classify plays in Rush Football based on a set of starting configurations and offensive and defensive plays [7]. They construct the rush football into a supervised learning problem and trained the classifier to learn an opponent's defensive plays from a set of spatiotemporal features.

Statistical and classification approaches are useful in the fact that they do not require extensive reasoning about another agent's actions. Statistical methods also adapt quickly to changes in the agent's strategy, and classifiers are often able to model underlying complex functions in an agent's behavior. However, these approaches also have drawbacks. Classifiers typically require a large dataset for training. Also, predictive models do not always generalize well to previously unseen opponents [?]. Additionally, a player's recent history may not always be a good predictor of playing style [?].

Uncertainty is also a large factor in playing poker. Southey et al. use a Bayesian probabilistic model to separate the uncertainty in game dynamics from the uncertainty in opponent type identification, while Bard and Bowling mitigate the game uncertainty by using a Rao-Blackwellized particle filter in poker to identify the opponent's state as well as a model of its playing dynamics [10].

2.2 Stereotyping

These are all ways of learning an opponent's strategy, but few scale well to a large number of agents. If the agent types are known, however, it is possible to fit them to a specific type. Teofilo et al. shosed that a Q-learning agent whose state incorporated predefined player types, which were calculated based on the frequency of past plays, outperformed basic/intermediate playing strategies. Felix and Reis automatically classify opponents based on tehir VP$IP (voluntary money in pot) and their AF (aggression factor) into four distinct categories: loose passive, loose aggressive, tight passive, and tight aggressive [11].

Lockett identifies several 'cardinal opponents', which are pre-defined models that describe all dimensions of a player's actions. They then use NEAT to evolve a 'Mixture Identifier', which classifies agents in terms of these cardinal opponents [8]. Particle filters have been also used in Fictitious Play to track switching between a fixed number of strategies represented as Hidden Markov Models.

These techniques focus on either building a model of each of your opponents, or relying on a set of predefined models. Modeling each of your opponents does not scale well when you have many opponents that need separate models for each. Relying on a set of predefined models gets around this problem, but wholly rejects the potential for adaptation of agents. We take a step away from these approaches, and instead use a simple identification of abilities rather than to predict actions. This allows the adaptation to be handled by the evolutionary framework rather than by adjusting models.

There are several current approaches to the implementation of the stereotyping concept. Stereotyping has been combined with the concept of 'trust' using a tree model to represent a stereotype, and these stereotypes are shared and updated by the agent community [2]. This is further developed by the concept of *stereotypical reputation*, which gives a mechanism by which agents that have no set opinion can use the opinions developed by others in the system through *stereotypes* [3]. Denzinger and Hamdan use stereotypes with a periodic reevaluation of the chosen stereotype, and may switch between different stereotypes [?]. Bard and Bowling present a method of learning robust responses to several player types learned offline, which provides implicit modeling rather than relying on an explicit model of an agent's actions [?].

2.3 NeuroEvolution

Neuroevolution is a guided search technique that applies evolutionary principles to neural networks in order to adapt them to a problem. Neural networks are used as function approximators where the scaled values of the state elements are given as inputs, and the outputs correspond to action elements. The general algorithm used in neuroevolution is given as:

1. Generate k new population members through mutation operations.

2. Obtain fitness values for each population member by

simulation.

3. Keep the k population members with the highest fitness.

This is the neuroevolution algorithm that we use in this work.

Neuroevolution has been used often in the pursuit domain, particularly using coevolution [**?**, **?**]. The pursuit game traditionally consists of four predators and one prey on a gridworld.

3. EXPERIMENTAL SETUP

Our experiments are performed on the n-agent pursuit game. This game is focused pursuers pursuing a prey agent. Traditionally the game is played with four agents pursuing one prey. However, because we wanted to test the scalability of our algorithm we extended the traditional approach in **??** to accommodate any number of agents and prey. We then vary this number in our subsequent experiments to test our approach's response to varying system dynamics.

3.1 N-Agent Pursuit Domain

The pursuit domain provides a testing domain that illustrates the potential benefit of stereotypes for minimal representation of complex interactions. We experiment with a heterogeneous n-player version of the pursuit game. This requires some deviations from the canonical pursuit game, but preserves the core interactions between the predator and prey dynamics. This demonstrates many advantages of our approach because there are many variations in the potential capabilities of collaborators. We can show on a well-explored problem how our approach improves collaboration, without many of the intricacies in the air traffic domain.

The pursuers are heterogeneous agents with the goal of capturing the prey, which are a homogeneous set of learning agents.

3.1.1 Grid Movement

Agents move on a square grid with an edge length $L = 50$. Each step in the simulation, an agent can choose to move in one of the four cardinal directions, up to the equivalent distance of two squares away, unless its type specifies otherwise. Simultaneously, prey select actions, moving a random distance and turn away from a predator. If a predator blocks the movement of a prey, the prey does not move. Similarly, if a predator or a prey attempt to move into a wall, the rotation is enacted but the movement is not.

3.1.2 Agent Representation

Often the pursuit domain is stated as a coevolution problem, with both predators and prey competing to develop better policies. In our approach we focus on the evolution of the predators and the development of coordination strategies.

We use neuro-evolution as the adaptation mechanism for the pursuit domain. Elements of the agent state scaled on the range $[0, 1]$ form the input to the neural network. The neural network then has 15 hidden units, and 2 outputs. The first output represents the amount by which the agent chooses to modify its current orientation, $d\theta$, scaled between $[0, 2\pi]$. The second output represents the movement length of the agent dL, which is scaled between $[0, l_{max}]$, where

Figure 1: A representation of the pursuit game.

$l_max = 2$ is the maximum value that an agent can choose to move.

An individual $\mathcal{I}$'s state is comprised of $5 + N_\tau$ different parameters if type information is included, where N_τ is the number of types defined in the system. The first parameter is the absolute orientation $o_\mathcal{I}$ of individual $\mathcal{I}$. The second and third are the angle and distance to the nearest prey, θ_{prey} and d_{prey} respectively. The fourth and fifth elements are the angle and distance to the nearest neighboring pursuer, θ_{NN} and d_{NN} respectively. The last state elements, which are optionally included based on the setting of the run, are represented by the binary vector $\tau_N N$, which identifies the type of nearest neighbor. These form a tuple $\{o_{me}, d_{prey}, \theta_{prey}, d_{NN}, \theta_{NN}, \hat{\tau}_{NN}\}$, which serves as the inputs to the neural network during the run. The distance elements are scaled by an observation radius $dlimit = 20$, and if the nearest prey or pursuer is outside of this limit it is assigned the maximum limit. The angular elements o_{me}, θ_{prey}, and θ_{NN} are scaled by 2π, unless the prey or pursuer is outside the observational radius, and then this value defaults to 0.

Evolution is performed by mutating a population of $k = 10$ neural networks. At each new generation, k new children are created by mutating the parent's weights with a probability $P_{mut} = 0.1$, which adds weights by adding a sample from a normal distribution with mean $\mu = 0$ and standard deviation $std = 1.0$. These $2k$ neural networks are then tested, and assigned a fitness based on the average of $N_{trials} = 20$ trials.

We calculate fitness by calculating the distance from the prey at each timestep. We calculate the elemental fitness of an individual $\mathcal{I}$ on trial run s:

$$TrialFitness(\mathcal{I}, s) = \begin{cases} t_{catch} & \text{if } t_{catch} \leq T \\ \sum_{t=1}^{T} \delta(\mathcal{I}, t) & \text{if } t_{catch} > T \end{cases}$$

where t_{catch} is the time required for the pursuer to catch the prey, $\delta(\mathcal{I}, t)$ is the distance of individual $\mathcal{I}$ from the nearest prey at time t. This is averaged across all trial runs to obtain the fitness of the individual for that epoch:

$$Fitness(\mathcal{I}) = \sum_{s=1}^{N_{trials}} TrialFitness(\mathcal{I}, s);$$

3.1.3 Types in the Pursuit Game

We model a set of pursuers with different limitations. These limitations are observable to other pursuers. We define four types of predators in the pursuit game:

- **Standard:** A standard-type pursuer has a full range of motion and may turn up to 2π radians per timestep. At each timestep it may also choose to move forward by a distance of up to l_{max}.
- **Slow-turning:** A slow-turning type may only turn a maximum of $\pi/2$ radians per timestep.
- **Fast:** The distance selection of a fast agent is scaled by 2. This means that a fast agent can move a maximum of $2l_{max}$ steps per timestep. Because this merely scales the output, fast-type pursuers retain the capability to move slower per timestep by reducing their neural network output accordingly.
- **Erratic:** An erratic-type pursuer evolves a neural controller as the other pursuers but it will with probability P_{defect} select a random angular and distance output not specified by the neural network. This has the same capabilities as a standard-type pursuer.

The introduction of heterogeneous types adds an additional dynamic to this domain. A slow-turning agent may not be able to maneuver well toward a prey, and may focus instead on better positioning in order to have the other agents catch the predator. Other pursuers may learn to avoid depending on capture operations with an erratic-type agent, as it may allow the prey to escape.

3.2 Evasion in the Pursuit Domain

Capturing prey with random movement provides a relatively easy learning problem for the pursuers, and capture is more likely achieved by chance. We also investigate pursuer response to intelligent prey movement. We investigate performance against three separate movement algorithms: a charged particle avoidance strategy, and learning prey.

4. RESULTS

4.1 Agent Coordination

To test whether our approach increases the ability of agents to coordinate with one another, we vary the parameters of the predator prey domain. The ratio of predators in this domain dictates whether it is a coordination problem or a congestion problem. In Figure 2 we can see that...

Figure 2: This shows the impact of coordination on the predator-prey domain.

In many cases the agent appears to get into the wrong area of the search space with agents and is not able to escape. For this reason, we train first without observing types, and then at epoch $t = 100$ in the simulation types become observable. This addition adds an array of nodes that observe the type of the agent, with randomly generated weights. Because the nodes are added near convergence of the simulation, the weights of the nodes are set to be $\frac{1}{10}$ the size of the weights created at initialization. We show the impact of this delay in Figure 3.

4.2 Prey Type Influence

The type of the prey can also influence the dynamics in the game. We experiment with several different prey types in order to determine the benefits of including a stereotype mechanism with a subtly different learning problem.

5. DISCUSSION

Using a neural network for a policy representation offers a mechanism for handling unnecessary information. If the type of the coordinating agent is not helpful, the weights simply decrease during the evolution process to reduce this noise.

One of the main insights that this gives us is a look into the nature of roles in the predator-prey domain. The nature of this game changes as the ratio of predators to prey changes. This is because the coordination aspect of the game is increased when there are many predators. It becomes much more important to finding the optimal solution to know *who* you are coordinating with, and a general outline of their capabilities.

Future work involves automatically identifying roles of agents in the system. Even when there are no differences in capability of agents, agents can learn heterogeneous policies

Figure 3: This shows the average capture times for several different prey evasion strategies.

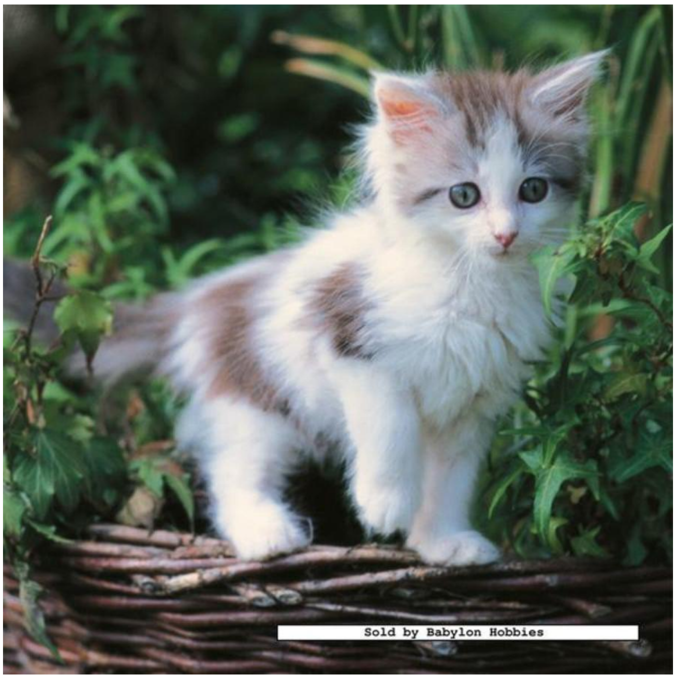

Figure 4: This shows the average capture times for several different prey evasion strategies.

in order to solve a model. If these can be characterized they can be used to steer toward better inter-agent coordination.

6. REFERENCES

[1] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI/IAAI*, pages 493–499, 1998.

[2] Chris Burnett, Timothy J Norman, and Katia Sycara. Bootstrapping trust evaluations through stereotypes. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 241–248. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[3] Chris Burnett, Timothy J Norman, and Katia Sycara. Stereotypical trust and bias in dynamic multiagent systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(2):26, 2013.

[4] David Carmel and Shaul Markovitch. Opponent modeling in multi-agent systems. In *Adaption and Learning in Multi-Agent Systems*, pages 40–52. Springer, 1996.

[5] Doran Chakraborty, Noa Agmon, and Peter Stone. Targeted opponent modeling of memory-bounded agents. In *Proceedings of the Adaptive Learning Agents Workshop (ALA)*, May 2013.

[6] Omer Ekmekci and Volkan Sirin. Learning strategies for opponent modeling in poker. In *AAAI-13 Workshop*, 2013.

[7] Kennard Laviers and Gita Sukthankar. A real-time opponent modeling system for rush football. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2476–2481. AAAI Press, 2011.

[8] Alan J Lockett, Charles L Chen, and Risto Miikkulainen. Evolving explicit opponent models in game playing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2106–2113. ACM, 2007.

[9] Marc Ponsen, Jan Ramon, Tom Croonenborghs, Kurt Driessens, and Karl Tuyls. Bayes-relational learning of opponent models from incomplete information in no-limit poker.

[10] Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*, 2012.

[11] Luís Filipe Teófilo, Nuno Passos, Luís Paulo Reis, and Henrique Lopes Cardoso. Adapting strategies to opponent models in incomplete information games: a reinforcement learning approach for poker. In *Autonomous and Intelligent Systems*, pages 220–227. Springer, 2012.

[12] Colin R Williams, Valentin Robu, Enrico H Gerding, and Nicholas R Jennings. Using gaussian processes to optimise concession in complex negotiations against unknown opponents.