Optimal Rewards for Cooperative Agents

Bingyao Liu, Satinder Singh, Richard L. Lewis, and Shiyin Qin

***Abstract*—Following work on designing optimal rewards for single agents, we define a multiagent optimal rewards problem (ORP) in cooperative (specifically, common-payoff or *team*) settings. This new problem solves for individual agent reward functions that guide agents to better overall team performance relative to teams in which all agents guide their behavior with the same given team-reward function. We present a multiagent architecture in which each agent learns good reward functions from experience using a gradient-based algorithm in addition to performing the usual task of planning good policies (except in this case with respect to the learned rather than the given reward function). Multiagency introduces the challenge of nonstationarity: because the agents learn simultaneously, each agent's reward-learning problem is nonstationary and interdependent on the other agents evolving reward functions. We demonstrate on two simple domains that the proposed architecture outperforms the conventional approach in which all the agents use the same given team-reward function (even when accounting for the resource overhead of the reward learning); that the learning algorithm performs stably despite the nonstationarity; and that learning individual reward functions can lead to better specialization of roles than is possible with shared reward, whether learned or given.**

***Index Terms*—Intrinsic motivation, multiagent learning, optimal rewards, reinforcement learning.**

I. Introduction

We consider the problem of designing reward functions for individual agents in multiagent sequential decision-making problems in the cooperative (specifically, common-payoff or *team*) setting. Throughout we will be focused on decentralized planning/learning approaches and will assume no direct communication among the agents. Furthermore, the shared reward function that defines the objective in the team problem is assumed known. The usual approach to such problems would, of course, make the given joint reward function the reward function for each agent. Why should a designer of multiagent systems do anything different? As we discuss below, in single agent settings it has been shown that designing reward functions different from the given reward function can help mitigate agent limitations, e.g., help overcome computational limitations that prevent perfect planning. In this paper, directly following work [1], [2] on designing optimal rewards in single agent settings, we present an algorithm for learning individual reward functions in multiagent team problems and investigate three questions: 1) whether multiagent optimal rewards are capable of overcoming agent limitations that include individual agents not knowing exactly how other agents would behave; 2) whether our proposed multiagent optimal reward function learning algorithm is able to successfully handle the nonstationarity that comes from multiple agents learning simultaneously when their actions collectively impact the experience of all agents; and 3) whether in some domains learning individually-customized reward functions leads to more effective specialization of roles for the agents, and thus more effective coordination among the agents.

Background on Designing Rewards in Single-Agent Settings. The problem of designing rewards for single agents is typically studied in cases where a reward function is initially *unavailable*.[1] Nevertheless, multiple approaches have been developed for the seemingly counterintuitive problem of designing reward functions when already given a reward function. For example, reward-shaping of the variety developed by Ng *et al.* [10] considered the question of what space of reward functions yields the same optimal policy as the given reward function with the hope that such a space contains an alternative reward function that makes the learning task faced by the agent easier. Shaping [11], as practiced in animal learning in Psychology commonly produces a sequence of reward functions or tasks that start easy and become harder with the hope that an animal can thus be led to performing well on the given (and ultimate in the sequence) reward function. More recently, intrinsic rewards based on psychological motivations such as curiosity and exploration that are added to the given reward as bonuses have been shown to improve the performance of computational agents [12]–[21].

In this paper, we exploit the recent optimal rewards framework of Singh *et al.* [1] that stems from the observation that reward functions play two distinct roles in autonomous agents: an *evaluation* role, because cumulative expected reward determines a preference-ordering over policies, and a *guidance* role, because approximate value functions computed/learned using

Manuscript received March 07, 2014; revised July 17, 2014; accepted September 08, 2014. Date of publication October 13, 2014; date of current version December 09, 2014 . This work was supported by NSF Grants IIS-0905146 and IIS-1148668. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

B. Liu is with the School of Automation Science & Engineering, Beihang University, Beijing, China (e-mail: bingyao.liu@gmail.com).

S. Singh is with the Division of Computer Science & Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: baveja@umich.edu).

R. Lewis is with the Department of Psychology, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: rickl@umich.edu).

S. Qin is with the School of Automation Science and Engineering, Beihang University, Beijing, China (e-mail: qsy@buaa.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAMD.2014.2362682

[1]For example, the field of preference elicitation develops methods for eliciting an approximate reward function from some human expert through queries (e.g., [3] and [4]), while the field of inverse reinforcement learning infers an approximate reward function from data observed by having a human expert teleoperate the autonomous agent (e.g., [5] and [6]). Of course, other approaches to dealing with unknown reward functions in constructing autonomous agents eschew the problem of designing rewards altogether and instead learn agent-policies directly from observed expert behavior using some form of imitation learning (e.g., [7]) or learning by demonstration (e.g., [8]) or supervised learning (e.g., [9]).

the reward function determine the actual behavior of the autonomous agent. Separating these two distinct roles into two separate reward functions (with the given reward function used for evaluation) sets up the formal **o**ptimal **r**ewards **p**roblem (ORP) whose solution is the reward function used for guidance (the ORP can also be seen to be an optimization approach to learning intrinsic reward functions [1]). If the agent is limited in some form (constraints on computation or knowledge), then solving the ORP for a guidance-reward function can improve the agent's performance relative to using the evaluation-reward function for guidance [2]; in other words, optimal rewards can mitigate agent limitations. In addition, algorithms and architectures to solve the ORP have demonstrated empirical success [22] and we will build upon those in this paper.

Our departure from prior work on ORP is in extending it to multiagent teams. Our contributions include: 1) defining the optimal rewards problem in the multiagent team setting; 2) extending a single-agent gradient-based approach to learning optimal rewards from experience to multiagent teams; 3) empirically demonstrating that despite the nonstationarity introduced by the multiagency, learning optimal rewards can mitigate agent limitations; and 4) empirically demonstrating that learning optimal rewards in teams can lead to more effective specialization in that each agent is able to learn its own guidance-reward whilst sharing the team's evaluation-reward.

II. Multiagent Optimal Rewards Problem

Let the set of agents be $\{A^i\}_{i=1}^n$; our mathematical formulation is for arbitrary n though our empirical evaluation will be restricted to $n = 2$. At time step t agent A^i gets an observation $o_t^i \in O^i$ from the environment M and takes an action $a_t^i \in A^i$. Agent i's history at time t is $h_t^i = o_1^i a_1^i \cdots o_{t-1}^i a_{t-1}^i o_t^i$. The joint observation, action and history at time t are denoted o_t, a_t and h_t respectively. The joint action a_t causes a stochastic change in the underlying state of the environment which in turn influences the next joint observation o_{t+1}. The given joint (i.e. common or team) reward function, hereafter the *objective reward function*, is denoted $R^{\mathcal{O}}$ and will be used to evaluate joint histories, i.e., the objective utility achieved by the agents after history h is $U^{\mathcal{O}}(h) = \frac{1}{|h|}\sum_{t=1}^{|h|} R^{\mathcal{O}}(h_t)$. In the conventional formulation the reward to each agent at history h would be $R^{\mathcal{O}}(h)$. Here we allow an individual *guidance reward function* R^i for agent A^i (here the notation of superscript i serves to both index the agent and denotes that the guidance reward function is **i**nternal to the agent). Thus the guidance utility for agent i with joint history h is $U^i(h) = \frac{1}{|h|}\sum_{t=1}^{|h|} R^i(h_t)$. (In practice reward functions, both objective and guidance, are defined as mappings from some abstraction of history to scalar rewards but for complete generality we present our formulation with no abstractions.) In single-agent settings, the agent's guidance reward function determines its behavior. In multiagent settings, the joint setting of the multiple individual guidance reward functions will collectively determine the team's behavior. Note that the agents cannot communicate with each other and thus any coordination among the agents has to be the result of independent learning/planning guided by the individual reward functions.

Agent A^i using guidance reward function R^i is denoted A^{R^i} for ease of notation.

Definition of multiagent optimal rewards problem. Given a set of agents $\{A^i\}_{i=1}^n$, a search space[2] of joint-reward functions $\{R^i\}_{i=1}^n$, an environment M, and the objective reward function $R^{\mathcal{O}}$, the set of reward functions $\{R^{i,*}\}_{i=1}^n$ is jointly optimal for the team of agents, if and only if

$$\{R^{i,*}\}_{i=1}^n = \underset{\{R^i\}_{i=1}^n \in \{R^i\}_{i=1}^n}{\arg\max} \mathbb{E}_{h\sim\langle\{A^{R^i}\}_{i=1}^n, M\rangle}[U^{\mathcal{O}}(h)] \quad (1)$$

where $\mathbb{E}$ is the expectation operator and indicates that the quantity to be maximized by the selection of $\{R^i\}_{i=1}^n$ is the *expected objective utility* $U^{\mathcal{O}}(h)$ of histories $h \sim \langle\{A^{R^i}\}_{i=1}^n, M\rangle$, that is, histories sampled[3] from the interaction of the set of agents $\{A^{R^i}\}_{i=1}^n$ using guidance rewards $\{R^i\}_{i=1}^n$ in environment M.

In other words, the optimal set of guidance reward functions for the team of agents is the choice of individual reward functions that guide the team of agents to joint-behavior that in expectation maximizes the objective utility as measured by the given joint (team) reward function. This paper is about approaches to solving this new multiagent ORP.

Multiagent optimal rewards versus other approaches to multiagent rewards. Crucially, the optimal set of guidance reward functions are optimal not just with respect to the environment and given objective reward function, but also with respect to the details of the architectures and algorithms of the various agents. Specifically, the agents and their limitations (including those arising from multiagency) help determine the distribution over h which in turn determines the effectiveness of choices of guidance reward functions. This sets the multiagent optimal rewards problem and its solutions apart from other approaches to designing rewards that also frame the problem of finding good rewards or incentives in multiagent settings as an optimization problem, e.g., from the collective intelligence approach [23] that ignores the limitations of the agents and focuses on dealing with strategic settings, and from approaches based on mechanism design [24] that typically assume the availability of a central auctioneer and more importantly assume that the agents know and can communicate their correct utility function to the central auctioneer. This paper is also different from a recent and related approach to learning social awareness via intrinsic rewards for multiagent systems [25] that only demonstrates weak-mitigation, while our emphasis is on the greater challenge of strong-mitigation (see Section III for this important distinction).

The potential for specialization. By definition, in team problems the agents have to learn to cooperate and coordinate to achieve high shared objective utility. In some problems such coordination requires different members of the team to adopt different roles, i.e., to specialize. Thus one of the questions of interest is whether the ability to learn different guidance reward functions by the different agents can lead to increased specialization of roles, and thereby an improvement in coordination among the team. We address this question empirically in Section V-B.

[2]Note that R is used to denote a space of reward functions while R is used to denote a specific reward function.

[3]Here the distribution over histories results from the stochastic effects of the joint action on the agent state as well as any stochasticity in the agents' action selection as a function of history.

III. Multiagent Strong-Mitigation Architecture

As in the single-agent setting, it is straightforward (see Eq. (1) that as long as the search space of reward functions for each agent contains the objective reward function, the objective utility obtained by using the optimal guidance reward functions will be at least as high as that obtained by the conventional use of the objective reward function as the guidance reward function for all agents. For the case of bounded agents, the reward space may contain guidance reward functions that yield *higher* objective utility than the objective reward function. This is called the *weak-mitigation* property [26] in single-agent settings, and it has a straightforward analog in multiagent settings as well. But note that the existence (in the reward space) of guidance reward functions better than the objective reward function, which is all that weak-mitigation demands, does not mean that such reward functions can be found cheaply or at all. A more interesting result would be to show that it is worth it for an agent to devote some of its limited computational resources to learning a good guidance reward function at the cost of having fewer computational resources to plan good behavior with respect to the guidance reward function, i.e. learning a guidance reward function and behavior policy simultaneously can still be beneficial even when accounting for computational costs. Such a favorable property was called *strong-mitigation*[26] in single agent settings and it accounts for the cost of finding good reward functions (while weak- mitigation ignores it). Our first experimental objective below is to demonstrate strong-mitigation in the multiagent team setting.

Bratman *et al.* [26] showed that strong-mitigation implicitly suggests a **n**ested **o**ptimal **r**eward and **c**ontrol (NORC) architecture in which an overall agent has two components, an *actor-agent* which is the usual agent that takes actions so as to optimize reward, and a novel *critic-agent* that learns guidance reward functions. They demonstrated strong-mitigation in a setting where the single-agent repeatedly plans from the current state to select a current action, and where the computational constraint is the CPU-time available per action-decision. For different constraints on the amount of CPU-time per decision available, they compared the performance of two agents: 1) an NORC agent that splits the available CPU-time between the critic-agent updating the guidance reward function and the actor-agent doing limited planning with respect to the guidance reward function learned by the critic-agent; and 2) a conventional agent that spends all available CPU-time per decision on planning more deeply. They showed that with limited CPU-time per decision, the NORC agent does better in terms of objective utility than the conventional agent. In this paper, we will follow the same empirical approach by considering limited CPU-time per decision constraints and comparing the performance of our multiagent NORC architecture (described next) and a conventional multiagent architecture.

Multiagent NORC Architecture. Fig. 1 illustrates our multiagent NORC architecture. Each agent has within it an actor-agent and a critic-agent. In our empirical work the actor-agent is a UCT-based planner (UCT stands for **U**pper **C**onfidence bounds on **T**rees) [27] and the critic-agent uses the **P**olicy **G**radient for **R**eward **D**esign or PGRD algorithm of Sorg *et al.* [22] (we describe this algorithm briefly below). The significant departure from the single-agent setting is that while each agent gets its own observation and produces its own action, it is the joint action of the agents that determines the shared objective reward.

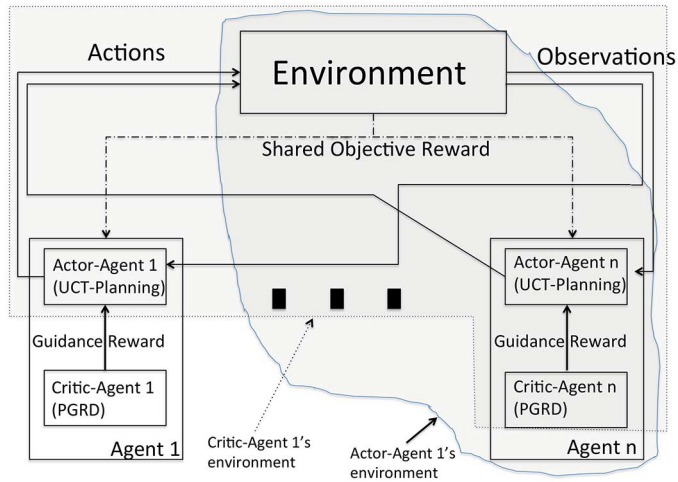


Fig. 1. Multiagent nested optimal reward and control (MNORC) architecture. Each agent is composed of a conventional actor-agent and a novel critic-agent. The critic-agent learns a guidance reward function that guides the associated actor-agent's behavior. The joint action of all the agents influences the shared objective reward received by all the agents. Note the large irregular shape showing that the actor–agent-1's environment includes the external environment as well as all the other agents, and the wider and more-rectangular shape showing that the critic–agent-1's environment includes its own actor–agent-1 as well as the external environment and the other agents.

Another way of understanding the additional challenge posed by multiagency is to consider the effective environment of a particular actor-agent. As shown in Fig. 1, the environment of actor–agent-1 includes not only the external environment but also all the other agents including their critic-agents. Even if all the other actor-agents are fixed planners, the fact that the critic-agents are learners and thus nonstationary make each actor-agent's environment nonstationary. Fig. 1 also shows a critic-agent's environment, and again because of multiagency it includes other learning components. The big open questions are whether multiple NORC agents simultaneously learning guidance reward functions and acting in the world will converge reliably and stably to good guidance functions for each agent, and whether multiagent strong-mitigation will be achieved, i.e., whether given CPU-time per decision constraints the multiagent NORC architecture will achieve more expected objective utility than the conventional multiagent architecture. Before we turn to our empirical results that are focused on these questions, we describe the actor-agent and critic-agent briefly.

Actor-Agents and the search for good policies. We assume that the actor-agents have a perfect model of the dynamics of the environment (but not what actions the other agent will take; see below) and use UCT to plan. UCT has two parameters T and D; it builds a search tree by simulating (from the current history) T trajectories of depth D where action-choices in the trajectory generation are treated as a bandit problem solved by the UCB1 [with **u**pper **c**onfidence **b**ounds (UCB)] algorithm [28] using the guidance reward function. The guidance reward function is

also used to compute from the resulting search-tree an estimated long-term utility for each action at the root node of the tree, and finally the best-estimated-utility action at the root node is selected. This process is repeated afresh at each time step for the current history as root node. The CPU-time per decision is heavily dependent on the two parameters T and D.

Because this is a distributed multiagent setting and the agents can not communicate with each other, there is an interesting challenge, however. When it builds a UCT tree, each agent must decide what actions the other agents would take at each step. A further complication is that if there is any learning by the agents, as will be the case if the guidance reward functions are learned through experience, then the behavior of the other agents will be nonstationary and thus difficult to predict. In all our experiments, agents learn the other agents stochastic policy using the empirical probabilities observed in historical data, and use this learned behavior model to sample the other agents actions during the tree-building process in UCT (we describe the learned models in more detail in Section V-D). Each NORC agent thus has two learning components, one for learning models of other agents behavior and one for learning guidance-reward functions.

Critic-Agents and the search for good guidance rewards. We assume that the search space of reward functions for each agent is defined as a continuously-parameterized function of reward-features (specific examples are in the empirical results below). The **p**olicy **g**radient for **r**eward **d**esign (PGRD) algorithm is based on the insight that the actor-agent is a procedure for translating the continuous parameters of the guidance reward function to a policy via planning. For certain classes of actor-agents, including the UCT agents used here, PGRD adapts standard policy-gradient approaches to use experience to update the guidance reward space parameters in the direction of the gradient of the expected objective utility. We refer the reader to Sorg *et al.* [22], [29] for the details of PGRD. In our multiagent NORC architecture each critic-agent implements a straightforward adaptation of the PGRD algorithm for the UCT-planning actor-agents. Specifically, multiagency requires that during UCT-planning each agent has to account for the action choices of all the other agents. As indicated above, our multiagent NORC architecture will use learned models of the other agents; we provide additional detail below in Section V-D. The main challenge that this adaptation of PGRD introduces is the fact that the additional nonstationarity in the learned other-agents-models may prevent convergence of the guidance reward function learning or create additional low-quality local minima in reward space.

IV. Experimental Objectives and Structure

We now describe a set of experiments in which we field our multiagent NORC architecture in two different environments, one (called "Food-Shelter") which has intuitively separate roles for the two agents, and one (called "Pursuit") which still demands cooperation between the two agents but does not have intuitively separate roles. We provide descriptions of the environments below, but first lay out what we intend to demonstrate with these experiments via a set of key comparisons among different agent types.

A. Overall Objectives and Key Comparisons

Our experiments have four main objectives:

1) demonstrate *strong-mitigation* in team problems using our multiagent NORC architecture;
2) demonstrate that learning individual guidance reward functions affords *better specialization* than using objective reward functions for shared guidance;
3) demonstrate that independent PGRD learning by the critic-agents can lead to *relatively stable and effective learning* of guidance reward functions;
4) demonstrate that even when the model of the other agent available for planning via PGRD is of poor quality, our multiagent NORC architecture is able to *outperform* the conventional use of objective reward functions for shared guidance.

These four demonstrations take the form of comparisons of the performance of three different multiagent systems (MASs) realized as multiagent NORC architectures that vary in the critic-agent:

- *UCT-ObjRe.* In this MAS, both UCT-based actor agents use the objective reward function as their guidance reward functions; i.e., the critic agent supplies a single stationary guidance reward that is fixed to be the objective reward. Note that this is the conventional MAS implemented here by specializing the NORC architecture of each agent so that the critic-agent simply passes on the objective reward as guidance reward without any changes.
- *UCT-PGRD.* In this MAS, the actor-agents plan with guidance reward functions learned independently by the critic-agents via PGRD[4] using gradients of the objective utility with respect to the coefficients on the reward-features. We describe the search space of reward functions below.
- *UCT-PGRDSame.* In this MAS, both agents use PGRD to update guidance reward functions, but are constrained to use the same learned guidance reward function. We implemented this constraint by averaging the gradients computed independently for each agent and updating both reward functions with the same average gradient. All parameters for the PGRD algorithm here are the same as with UCT-PGRD agents.

To demonstrate strong mitigation (Objective 1), we must show that *UCT-PGRD* outperforms *UCT-ObjRe* with equivalent resource consumption. To demonstrate better specialization via learned individual guidance rewards (Objective 2), we must show that *UCT-PGRD* outperforms *UCT-PGRDSame* (the MAS constrained to learn a single guidance reward for both agents) and does so via increased specialization of reward and behavior. To demonstrate stable and effective learning of guidance reward functions (Objective 3), we must show that *UCT-PGRD* both outperforms *UCT-ObjRe* and does so while settling on stable guidance rewards despite the nonstationarity of the reward-learning problem. To demonstrate relative robustness of our architecture to the quality of the learned other-agent-model (Objective 4), we must show that

[4]Throughout all the experiments, we used the following PGRD parameters without tuning, discount rate $\gamma = 0.95$, temperature $\tau = 100$, learning rate $\alpha = 1e-4$, decay parameter $\beta = 0.9$, and regularization parameter $\lambda = 0$. See Sorg *et al.* [22], [29] for details about the role of these parameters.

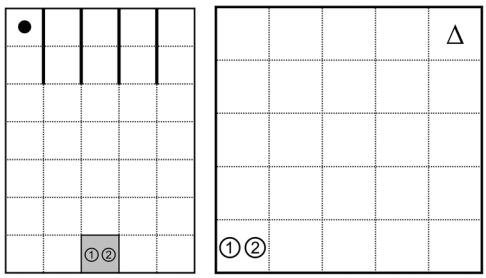


Fig. 2. (a) Food-Shelter. (b) Pursuit. Two multiagent domains. In each domain circled numbers indicate different agents. (a) In Food-Shelter, thick black lines indicate impassable walls, the large solid circle represents one possible location for the consumable food, and the shaded grid square in the middle of the bottom row is the shelter for the agents. (b) In Pursuit, the triangle represents the prey. Agents capture prey by pinning it in a corner. The configurations shows for each domain are the starting configurations.

UCT-PGRD outperforms *UCT-ObjRe* for multiple approximate models generated by a systematic exploration/generation of model quality.

B. Two Environments: Food-Shelter and Pursuit

Food-Shelter. The first environment has two clear and separate roles for the two agents: one should gather food and the other should keep the shelter repaired. As illustrated in Fig. 2(a), it is a 7×5 grid world with a shelter for the agents in the middle of the bottom row and 5 vertical corridors separated by impassable walls at the top.

Dynamics. The shelter breaks down (perhaps by unmodeled invaders) with probability 0.3 at each time step; once broken it remains so until repaired. Food appears at the top of one of the corridors; when food is gathered new food reappears at the top of a different (randomly chosen) corridor. The agents have four deterministic actions corresponding to each cardinal direction; if the resulting direction is blocked by a wall or the boundary, the action results in no movement. The agents have a *gather* action when at a food location and a *repair* action when at the shelter location that repairs a broken shelter immediately. The agents act simultaneously at each time step. The starting location of both agents is at the shelter.[5]

Objective reward. Gathering food contributes a joint objective reward of 1.0 while if the shelter is broken it contributes an objective reward of -0.1; otherwise the objective reward is 0. The agents objective utility is the expected objective reward per time step. In the results reported below we measure the objective utility over a lifetime of 5000 time steps. Thus the task for the two agents is to gather as much food as possible while keeping the shelter repaired; ideally one agent should focus on keeping the shelter repaired and the other agent should focus on gathering food.

Observations and prior-knowledge. Both agents observe the full state of the environment, i.e., observe the location of both agents, the location of the food, and whether the shelter is broken or not. The agents have a perfect model of the dynamics of the environment. They don't know how the other agent will act, of course, but they can observe how the other agent has acted in the past.

Pursuit. This second environment also demands cooperation between two agents to catch a prey but does not have intuitively separate roles for the two agents. As shown in Fig. 2(b), it is a 5×5 discrete, square grid-like world in which two agents (circles) need to coordinate to capture a prey (triangle). The prey's movements are fixed and not learned (see details below). The agents capture prey by pinning it any one of the 4 corners.

Dynamics. Both agents and the prey have five deterministic actions corresponding to each cardinal direction and the "stay in place" action; if the resulting direction is blocked by the boundary, the action results in no movement. At each time step, the two agents move simultaneously while the prey observes the agents new locations before moving. The prey moves so as to maximize distance to both agents.[6] The initial positions of the agents and prey are at opposite corners as shown in Fig. 2(b); the predators and prey are reset to this position once the prey is captured.[7]

Objective reward. The objective reward function $R^{\mathcal{O}}$ provides a reward of 1.0 when the prey is captured and a reward of 0 otherwise. The agents objective utility is the expected objective reward per time step. In the results reported below we measure the objective utility over a lifetime of 10,000 time steps.

Observations and prior-knowledge. Both agents observe the full state, i.e., the locations of the two agents and the prey. The actor-agents have a perfect model of environment dynamics. As for the Food-Shelter domain, the agents don't know how the other agent will act but can observe how the other agent has acted in the past.

C. Learning a Model for the Other Agent's Actions

In order to plan its actions using UCT, each agent has to account for the action choices of the other agent during its planning procedure. For this purpose, in our MNORC architecture, each agent incrementally learns a model of the other agent as the historical empirical probability of the other agent's actions conditioned on the observation of joint state in the environment, and uses this learned behavior model to sample the other agent's action during the tree building process in UCT. By allowing the empirical probability distribution over actions to condition on various subsets of the joint state, we can get different models with varying degrees of size and detail. Recall that because the other agent's behavior is nonstationary due to its learning of guidance reward functions, no model would be perfect. Specifically, for the results presented in Sections V-A to Section V-C, the model used for the Food-Shelter domain conditions learned

[5]Experimental results not reported here show that all our qualitative results are unchanged by setting agents to random initial locations; there is some additional variance in the expected utility in the early stages of learning but this does not impact the performance after the early stages.

[6]Formally, the prey chooses its action from the intersection of two sets: the first set is $a = \arg\max_{a \in A}\{\min[dist(l_1, \Delta'), dist(l_2, \Delta')]\}$, which maximize the distance to the closest predator agent; and the second set is $a = \arg\max_{a \in A}\{sum[dist(l_1, \Delta'), dist(l_2, \Delta')]\}$, which can further prevent the prey from moving towards the other predator agent; where l_1, l_2 represent the predator agents' locations and Δ' is the prey's next location after taking its action a, and $dist$ measures the Manhattan distance between the predator agent and prey.

[7]As for Food-Shelter, unreported experimental results on Pursuit show that our qualitative results are unchanged by setting/resetting agents to random initial locations.

action probabilities on the location of the other agent (teammate) only, while the model used for the Pursuit domain conditions learned action probabilities on the joint state, i.e., on location of the agent itself, location of the other agent, and the location of prey. These particular models were chosen because they performed best from among a set of choices; in Section V-D, we explore the robustness of our results to model quality and therein we describe multiple choices for models and their empirical consequences.

D. Search Space of Guidance Reward Functions

For both domains and both UCT-PGRD and UCT-PGRD-Same, we define the reward function search space as linear combinations of two features of history, the objective reward and an inverse-recency feature. The inverse-recency feature for agent A^i in history h^i, is defined as $\phi^i_{\text{inv.rec}}(h^i) = 1 - \frac{1}{c(h^i)}$, where $c(h^i)$ is the number of time steps since the agent A^i was in the same location as the location at the end of history h^i. Low values of this feature (close to 0) indicate that the agent is visiting a location it visited recently while large values of this feature (close to 1) indicate that the agent is visiting a location it has not visited recently. Formally, guidance reward as a function of history h is of the form

$$R^i(h) = R^{\mathcal{O}}(h) + \theta^i_{\text{inv.rec}}\phi^i_{\text{inv.rec}}(h^i) \tag{2}$$

where $\theta^i_{\text{inv.rec}}$ is the one continuous scalar parameter and its range provides the search space of reward functions.[8] Note that a positive value of $\theta^i_{\text{inv.rec}}$ will mean that the agent should want to visit locations not visited recently while a negative value of $\theta^i_{\text{inv.rec}}$ will mean that the agent should want to avoid locations not visited recently. This form of reward function space was used in previous work with single-agent ORP [22], [26] and was found to be effective in overcoming limited planning by encouraging a more systematic approach to exploration. We use this same reward function space because it is not task-specific and because seeking or avoiding exploration is an interesting abstract form of specialization that might manifest itself in interesting ways (and we will see such a phenomenon in Food-Shelter).

Eq. (2) defines a one dimensional continuous search space for PGRD in each critic-agent in the UCT-PGRD MAS for a 2-dimensional joint search space. Recall that for UCT-PGRD-Same the two critic-agents are required to use the same reward function and thus the joint search space is one dimensional. In both MASs, for $i = 1, 2$, the initial value of $\theta^i_{\text{inv.rec}} = 0$, i.e., the critic-agents start with the objective reward function as the guidance reward function.

V. Experimental Results

The results below are organized around the four main empirical objectives set out in Section IV-A and use the following methodology. For demonstrating strong mitigation we focus on computational time as the resource of interest. Because the majority of CPU-time-per-decision of an agent is spent on building a UCT search tree with T trajectories of depth D, instead of imposing a priori CPU-time-per-decision constraints, for both domains we evaluate all three MASs with UCT planning depths $D \in \{2, 3, 4, 5, 6, 7, 8\}$ and number of Monte Carlo (MC) trajectories $T \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$. By choosing these parameters, we densely explored a certain range of the CPU-time-per-decision of the agents in terms of D and T; from building a shallow-sparse search tree (e.g., $D = 2, T = 50$) to building a deeper-dense search tree (e.g., $D = 8, T = 500$). The aspects of MAS performance collected will include CPU-time per decision, average objective utility, the guidance reward functions learned, and the behavior of the agents. In particular, this means that we present the results in a way that does not commit to a particular combination of D and T as being best for some given time-per-decision limit. Analysis of these results will allow us to address our empirical objectives set out in Section IV-A.

A. Demonstrations of Strong Mitigation (Objective 1)

Food-Shelter Environment. Fig. 3(a) shows the per-time-step objective utility in Food-Shelter over 5000 time-steps plotted against CPU-time per decision for the three different MASs; the results are averages over 100 trials. The points in the figure correspond to different architectures (shown as circles for ObjRe, squares for PGRD, and stars for PGRDSame); different points of the same shape correspond to different UCT parameters (depth,trajectories). For all constraints on CPU-time per decision, using PGRD to learn reward functions is better than simply using the given objective reward. This is seen most clearly via the upper envelope curves in which UCT-PGRD—learning separate guidance reward functions for both agents—does the best, UCT-PGRDSame with the constraint that both agents use the same learned guidance reward functions is slightly worse, and finally UCT-ObjRe that employs the objective reward function for guidance is far worse.

Fig. 3(a) provides clear evidence of strong mitigation in this domain. Consider any point on the x-axis—for example 0.8 seconds per decision. If that were to be the computational bound on the MAS, spending some of those resources on learning guidance reward functions is far better than spending all those resources on action-planning. This improvement is the vertical gap between the corresponding upper-envelope curves at the vertical line $x = 0.8$. In other words, using PGRD in the critic-agent to learn guidance reward functions helps overcome the bounded depth/trajectories in UCT planning as well as the limited knowledge of the other agent's actions.

Fig. 3(b) shows the Depth-View of the same data in which each curve is for a specific depth D, with the points along a curve from left to right corresponding to larger number of trajectories T. Fig. 3(c) shows the Trajectory-View of the same data in which each curve is for a specific number of trajectories, with the points along a curve from left to right corresponding to larger depth D. Both views illustrate that the UCT-PGRD MAS significantly outperforms the UCT-ObjRe MAS. It is also evident that over the range of depth and number of trajectories

[8]Note that the inverse-recency reward function feature is only a function of h^i and not h and so does not require any communication among the agents. The objective reward is already assumed to be shared among the agents.

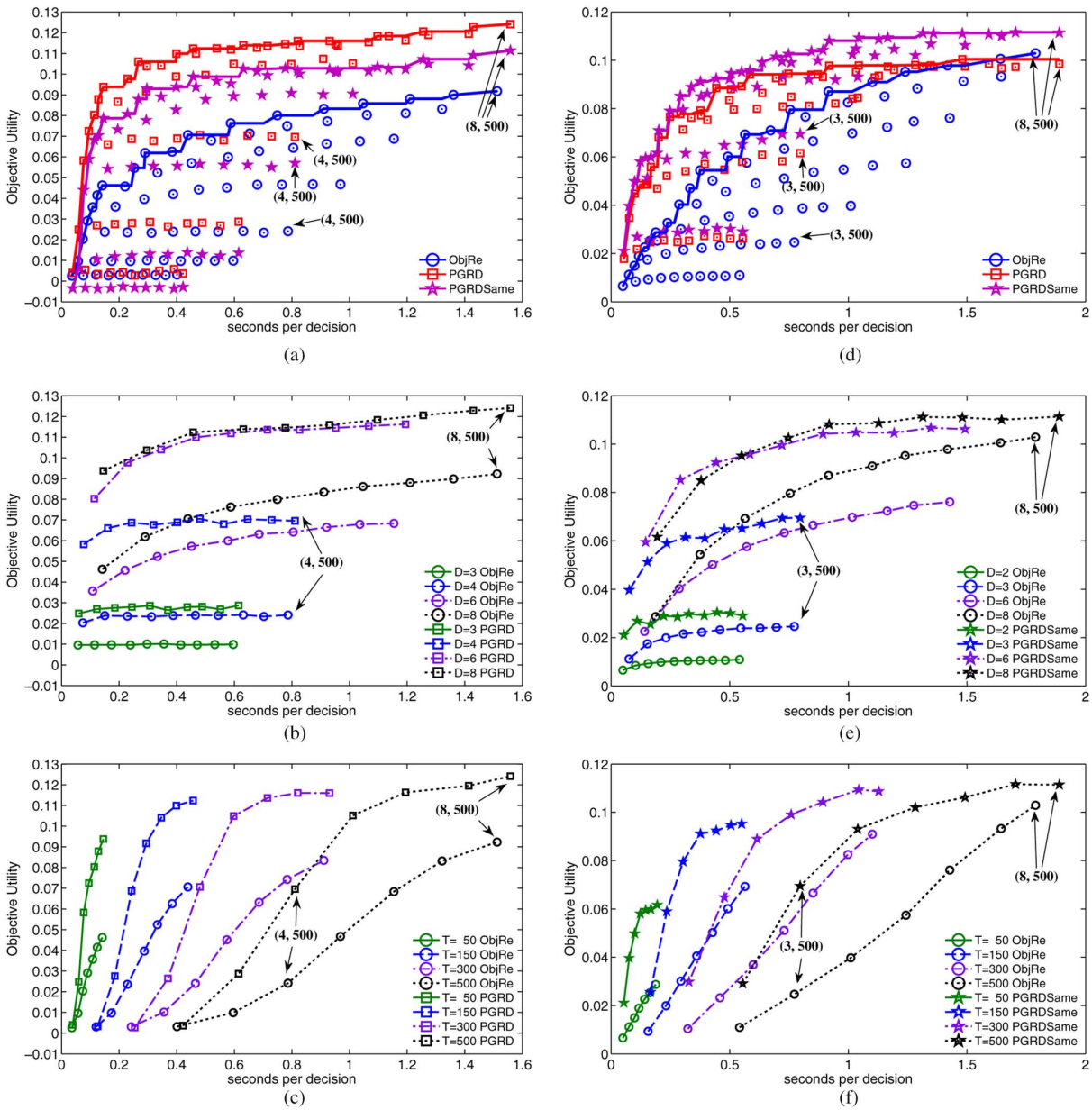


Fig. 3. (a) Food-Shelter: Envelope View. (b) Food-Shelter: Depth View. (c) Food-Shelter: Trajectory View. (d) Pursuit: Envelope View. (e) Pursuit: Depth View. (f) Pursuit: Trajectory View. Evidence for strong mitigation. Panels (a)–(c) present three different views of the same data for the Food-Shelter domain while Panels (d)–(f) present results similarly for the Pursuit domain. The x-axis in each figure is seconds-per-decision while the y-axis is average objective utility. In the *Envelope View* (top), each point corresponds to a particular choice of UCT parameters $(D = \text{depth}, T = \text{trajectories})$. The minimum point in this set for each MAS is $(2, 50)$ and the maximum is (8500). (To illustrate, two specific settings of (D, T) are labeled on the graph). The curves plotted in the Envelope views are upper-envelopes—that is, the best performing (D, T) at each seconds-per-decision resource level. Each curve corresponds to a different MAS architecture. In Depth View, each curve correspond to a particular planning depth D for each MAS, the marked points on a curve from left to right correspond to increasing number of trajectories. In Trajectory View, each curve correspond to a particular number of sampling trajectories T for each MAS, the marked points on a curve from left to right correspond to increasing planning depth. Note that the data points underlying the results in the three views for a domain are the same, only connected by curves with different meanings, and where for readability we omit some data points in the depth and trajectory views.

explored, limited depth is a far more consequential limitation on the agents than is limited number of trajectories because the slope of the curves are flatter in the Depth-View relative to the slope of the curves in the Trajectory-View.

Pursuit Environment. Fig. 3(d) shows the results for Pursuit in the same format as above. As was the case for the Food-Shelter, for all constraints on time per decision, using PGRD to learn guidance reward functions is better than simply using the given objective reward. The main difference from the Food-Shelter results is that the performance of the UCT-PGRDSame MAS is better than the performance of the UCT-PGRD MAS (this difference is unsurprising because Food-Shelter was designed to elicit specialization of roles which is precluded by PGRD-Same while Pursuit was designed to not need specialization of roles and hence PGRDSame has an easier guidance-reward-function learning task). We emphasize that as for Food-Shelter both PGRD-based MASs are significantly better than the UCT-ObjRe MAS. Strong-mitigation is observed again in that for all

constraints on the available seconds per decision, i.e., for all points on the x-axis, the vertical gap between the upper-envelope curves for UCT-PGRDSame and UCT-ObjRe is the performance advantage of spending some of the fixed computational resource on updating the guidance reward function.

Fig. 3(e) shows the Depth-View of the same data in which each curve is for a specific depth, with the points along a curve from left to right corresponding to larger number of trajectories. Fig. 3(f) shows the Trajectory-View of the same data in which each curve is for a specific number of trajectories, with the points along a curve from left to right corresponding to larger depth. In both views UCT-PGRDSame significantly outperforms UCT-ObjRe. As for the Food-Shelter domain, again it is evident that limited depth is a more consequential limitation on the agents than is limited number of trajectories.

B. Demonstrations of Specialization (Objective 2)

As expected intuitively from the specifics of the two domains, we did not find specialization in Pursuit and so focus here primarily on analyzing the results of Food-Shelter. Our first positive result concerning specialization is in the dominance of UCT-PGRD over UCT-PGRDSame in Food-Shelter as described above (see Fig. 3(a)): learning individual guidance rewards does produce greater to slightly greater performance, depending on the CPU-time per decision constraint, than learning a shared reward. But is this dominance in performance associated with clearly specialized *behavior* and clearly specialized *individual rewards?* We address these two questions here.

The expression of specialization in behavior. By design, in order to achieve high objective utility in Food-Shelter, one agent needs to focus on shelter-repairing work while the other focuses on food-gathering work. We defined a measure of this specialization as follows. For each agent we compute the following fraction: the number of times it gathers food divided by the number of times it gathers food and repairs shelter. For each run, the absolute value of the difference between this fraction for the two agents is a measure of the specialization for that run. This difference in specialization ratio is between 0 and 1. The more specialized the agents are the closer the specialization ratio will be to one, and the less specialized they are the closer the specialization ratio will be to zero. Fig. 4 shows the average amount of specialization as a function of CPU-time-per-decision in curves that correspond to different number of trajectories (the UCT parameter). The points along a curve from left to right correspond to increasing depth. It is clear that the UCT-PGRD MAS specializes far more than the UCT-ObjRe MAS. Finally, observe that in each pair of curves corresponding to the same number of trajectories, the rightmost points that correspond to the highest depths show that as the consequential computational limitation that small depth imposes on UCT is removed from the agents, the UCT-ObjRe MAS specializes just as well or even better than the UCT-PGRD MAS. This is because learning guidance reward functions can really only help in bounded agents and at the largest depth the agents have no planning bounds. In Pursuit, there is little specialization between the two agents' behavior, and this is explored further during the later discussion of the expression of specialization in the learned reward functions.

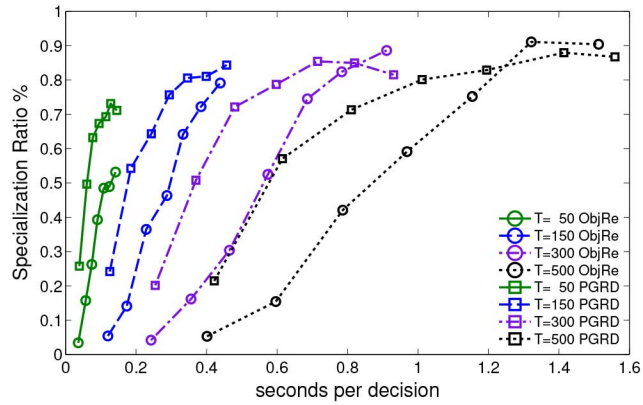


Fig. 4. Specialization Ratio Curves for Food-Shelter. Each point is for a distinct MAS (circles for ObjRe and squares for PGRD) using particular UCT parameters. The x and y values of each point are the seconds per decision and the specialization ratio respectively for the corresponding MAS. Curves connect points for MASs with the same number of trajectories in UCT. The main result is that PGRD agents achieve higher specialization ratios relative to ObjRe agents when both are constrained to similar CPU-time per decision. See text for further explanation and discussion.

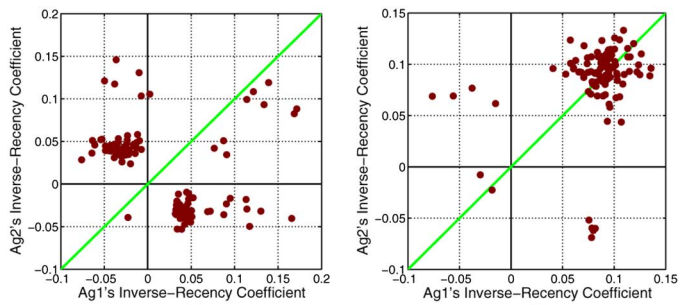


Fig. 5. The two panels (a) and (b) correspond to UCT-PGRD MAS ($D = 8, T = 500$) agents in Food-Shelter and Pursuit domains respectively. Each of 100 data points in panel (a) corresponds to a separate run and plots the coefficient for the inverse-recency reward feature for one agent against the same coefficient for the other agent. The striking specialization of reward functions in Food-Shelter is apparent from most of the points being off-diagonal in panel (a). This is in contrast to Pursuit in which the lack of reward function specialization is apparent in most of the points being close to the diagonal in panel (b).

The expression of specialization in the learned reward functions. What form did the learned guidance reward functions actually take in the two domains? Fig. 5(a) present the results for the UCT-PGRD MAS (with $D = 8$ and $T = 500$) on Food-Shelter. Each point is for a separate run; it plots the learned coefficients on the inverse-recency reward feature for the two agents. As is visually clear, one agent tends to learn a slightly negative coefficient while the other agent tends to learn a slightly positive coefficient. Recall that a positive coefficient for the inverse-recency reward features encourages the agent to explore, to visit places it hasn't visited recently, and this helps this agent become better at finding food which moves from place to place. The agent with a slightly negative coefficient for the inverse-recency reward feature is discouraged from exploring and this helps it stay put at the shelter location. Thus, it is clear that the critic-agents for the two agents learn quite different guidance reward functions, i.e., there is specialization in the guidance reward functions. Fig. 6(a–d) show that there is specialization also in behavior as a result. Each panel shows the number of time

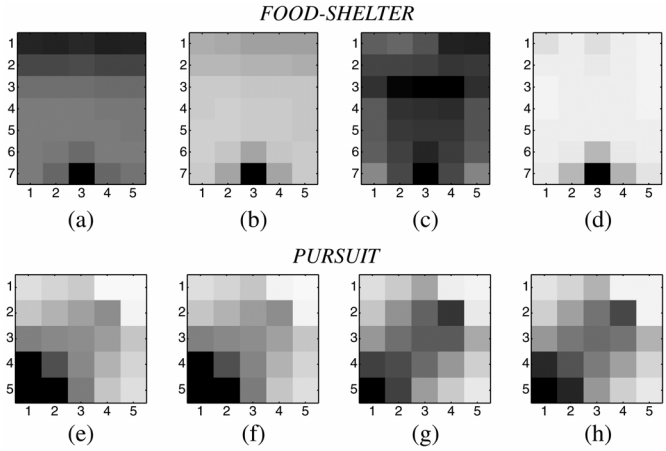


Fig. 6. (a) ObjRe Food Agent (b) ObjRe Shelter Agent (c) PGRD Food Agent (d) PGRD Shelter Agent (e) ObjRe Agent 1 (f) ObjRe Agent 2 (g) PGRD Agent 1 (h) PGRD Agent 2.The occupancy-maps above show the number of time steps an agent spends in each location on average over a 100 runs. Darker colors in grid-squares indicate more time spent in those locations. The top row is for Food-Shelter with UCT-parameters $D = 3, T = 500$. In the bottom row for Pursuit, the UCT-parameters are $D = 5, T = 500$. PGRD Agent 1 is the agent with larger coefficient for inverse-recency reward feature; PGRD Agent 2 has the smaller coefficient. The top-row provides evidence for greater specialization in behavior for Food-Shelter relative to the undifferentiated occupancy-maps in the lower-row for Pursuit. See text for further discussion.

steps spent in each location of the grid where darker color means more time spent. Panels (a) and (b) are for the UCT-ObjRe MAS and correspond respectively to the agent that gathers food more often and the agent that repairs the shelter more often. Similarly, panels (c) and (d) are for the UCT-PGRD MAS food and shelter agents. It is clear that there is increased behavioral specialization in the MAS that learns guidance reward functions; the shelter agent concentrates more on the shelter location in the UCT-PGRD MAS compared to the shelter agent in UCT-ObjRe MAS.

Fig. 5(b) plots the coefficients for the inverse-recency reward features for the two agents in UCT-PGRD MAS for Pursuit. Visually it is clear that for most of the 100 runs, agents learn roughly the same coefficients. There is little specialization of the guidance reward functions in Pursuit. This lack of behavioral specialization is apparent as well from panels (e–h) in Fig. 6 in which both UCT-ObjRe agent's occupancy-maps (panels e&f) look similar to each other as well as similar to the occupancy-maps for the two agents in the UCT-PGRD MAS (panels g&h).

The effect of enlarging the action space. In our final demonstration of specialization, we explored the effect of enlarging the action space in the Food-Shelter domain by adding a *Stay* action, which intuitively might be more usefully exploited by the Shelter agent than the Food agent (indeed its use would hurt the Food agent). But increasing the set of actions that are always available to each agent (recall that the *Gather* and *Repair* actions are available only when the agent is at a food or shelter location) from four (*North, South, East, West*) to five (*North, South, East, West, Stay*) also increases the branching factor in the planning search tree and so may make planning less effective for bounded agents. Thus, for any given fixed computational resource for the UCT planners (setting of depth D and trajectories T), adding the *Stay* action could either increase or decrease the obtained utility.

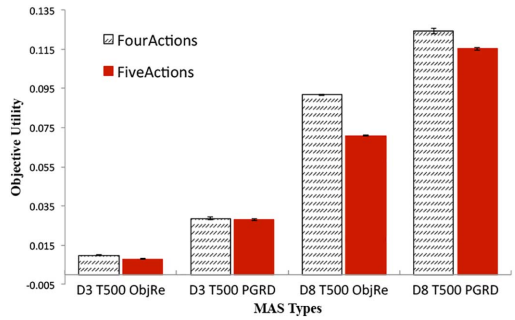


Fig. 7. Effects of adding a fifth action (*Stay*) to Food-Shelter. The x-axis labels identify the type of MAS (including the UCT parameters); the y-axis is objective utility. For each MAS type, the two bars correspond to the final (at end of lifetime) performance with FourActions and FiveActions respectively. The results presented are averages over 100 independent runs; standard error bars are shown. The smaller gap between FourActions and FiveActions for the PGRD agents relative to the ObjRe agents shows how PGRD is able to successfully mitigate the additional planning burden caused by the fifth action. We discuss in the main text how this mitigation is achieved by exploiting this fifth action for more effective specialization.

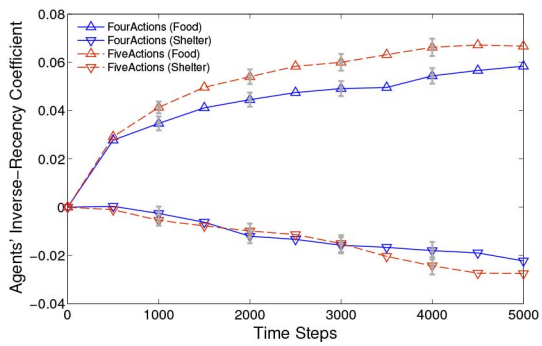


Fig. 8. Evidence that the addition of the fifth navigation action (*Stay*) increases specialization of the guidance reward functions. The graph shows reward learning curves for the PGRD agent ($D = 8, T = 500$) with different action sets. Curves marked with Δ represent the agent learning a larger/positive coefficient on inverse-recency feature (indicating a Food-agent), and the curves marked with ∇ represent the agent learning a smaller/negative coefficient (indicating a Shelter-agent). Each curve is averaged over 100 independent runs.

The interesting questions we ask here are whether the UCT-PGRD agents will better exploit the additional action and mitigate the additional planning cost, and whether they will do so through specialization of the reward function. The answer to both questions is yes. Fig. 7 provides evidence that there is a planning cost incurred by all agents by adding the *Stay* action, but that this cost is mitigated by the PGRD agents. Fig. 8 shows the learned reward features over time for the original four-navigation-action setting and the new setting with the *Stay* action. There is evidence that the additional action increases reward function specialization—there is additional separation of the two agents' reward functions in the five-action setting. There is also a difference in behavioral specialization between the PGRD and ObjRe agents: the number of Stay actions executed by PGRD Food-Agents is far smaller than the ObjRe agents (185 vs. 1113 for $(D = 3, T = 500)$ and 140 vs. 683 for $(D = 8, T = 500)$). Thus, the ObjRe agents are obtaining less food because the Food-Agents are spending more time on useless *Stay* actions.

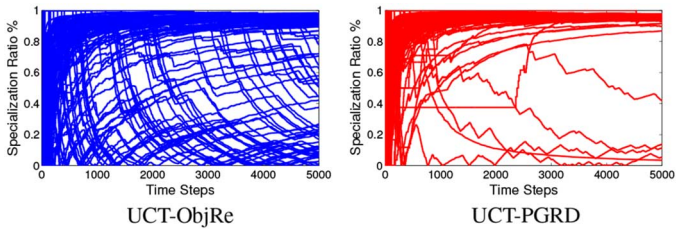


Fig. 9. Stability of learning guidance reward functions. The graphs show the specialization ratio as a function of time steps for 100 independent runs in Food-Shelter for UCT-ObjRe and UCT-PGRD MASs (with $D = 5, T = 500$). See text for more details and discussion.

In summary, there is evidence that PGRD mitigates the additional planning cost incurred by adding an action that is useful to one agent but not another; there is evidence from the learned reward features that enlarging the action space in this way increases the reward function specialization; and there is evidence that the mitigation arises because one agent learns to exploit the action while the other learns to more effectively ignore it.

C. Demonstration of Stability in the Face of Nonstationarity (Objective 3)

Fig. 9 illustrates how adapting the guidance reward functions can lead to more stable agent behavior. The curves in the two panels show the specialization ratio as a function of time for 100 different independent runs in Food-Shelter (we do not present results for Pursuit because by design the agents do not specialize in that domain). The left panel is for UCT-ObjRe and the right panel is for UCT-PGRD. It is apparent that for a greater fraction of runs in UCT-PGRD the specialization ratio converges to nearly 1. This is evidence of behavioral stability in that relative to UCT-ObjRe in UCT-PGRD it is more often the case across runs that one agent mostly focuses on food whilst the other focuses mostly on shelter.

D. Demonstration of Ability to Improve Performance with Approximate Models (Objective 4)

In the results reported thus far on Food-Shelter all of the architectures used what we will call Model1 below, while all the results reported thus far on Pursuit used what we will call Model 6 below. These choices were determined via exploration of seven models (Model0 to Model6), described next.

All the models defined below share the following common structure. Each agent independently learns a separate model of the other agent. At any given time step, each agent's model is a mapping from some subset of the full joint state to a probability distribution over the other agent's actions. These mappings are all initialized to be uniformly random and then are updated to be the empirical probabilities from the observed history; thus the model changes from time step to time step. When planning the action to execute at a time step, each agent uses the learned model's mapping to sample the other agent's actions during the tree building process in UCT. The actions taken by the two agents at a time step add to the history available at the next time step, and this cycle repeats until the end of the lifetime. So what is different across the different models? What differs is

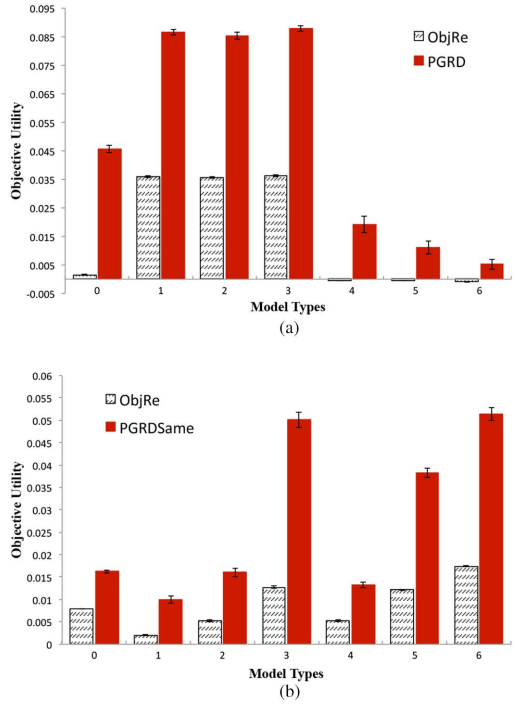


Fig. 10. (a) Food-Shelter. (b) Pursuit. Performance with seven different models in Food-Shelter and Pursuit. Panel (a) corresponds to Food-Shelter with UCT-ObjRe MAS and UCT-PGRD MAS (with UCT-parameters $D = 5, T = 100$ and 5000 time steps). Panel (b) corresponds to Pursuit with UCT-ObjRe MAS and UCT-PGRDSame MAS (with UCT-parameters $D = 3, T = 100$ and 10000 time steps). The x-axis is the label of the model-type used while the y-axis is averaged objective utility. The two bars within the same model-type label represent the averaged objective utility achieved by UCT-ObjRe MAS and UCT-PGRD MAS in Food-Shelter, and UCT-ObjRe MAS and UCT-PGRD-Same MAS in Pursuit. The results presented are for averages over 100 independent runs.

the subset of state variables that form the domain of the mapping learned in constructing the model. We define these subsets next.

Recall that for the Pursuit domain the full joint state at any given point of time is composed of three state variables: the location of the modeling agent (*locSelf* below), the location of the other agent (*locTeammate* below), and the location of the prey (*locFood* below). For the Food-Shelter domain, in addition to the same three state variables there is a fourth state variable, the status (whether broken or repaired) of the shelter; however, we found that including this variable in the domain of the model's mapping was not useful and hence we do not consider this variable below. Specifically, we evaluated the relative performance of the different MAS architectures for each of the following seven models:

- Model0: empty (unconditional distribution);
- Model1: locTeammate;
- Model2: locFood;
- Model3: locTeammate, locFood;
- Model4: locSelf;
- Model5: locSelf, locFood;
- Model6: locSelf, locTeammate, locFood.

Fig. 10 shows the end-of-lifetime objective utility per time step in Food-Shelter (Panel (a)) & Pursuit (Panel (b)) plotted as a bar-graph for each of the seven models above. For each

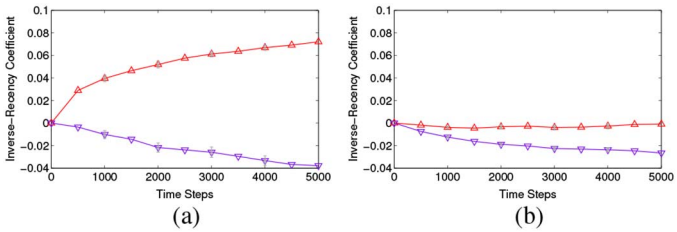


Fig. 11. Learning curves (of reward parameter) for the two agents in UCT-PGRD MAS in Food-Shelter domain using UCT-parameters $D = 5, T = 100$. Panel (a) is for the case when the two agents use Model1 while Panel (b) is for the case when both agents use Model6. The x-axis in each panel is time steps while the y-axis is the value of inverse-recency coefficient (see Eq. (2)). Each (learning) curve thus shows the adaptation of the inverse-recency coefficient with time steps. The results are an average over 100 independent runs in which the up-pointing triangles are averages for the agent with the larger coefficients and the down-pointing triangles are averages for the agent with smaller coefficients. Using Model1 allows for a clear separation between the two agents in that one of the agents quickly learns to use a positive coefficient and thus rewards exploration while the other agent quickly learns to use a negative coefficient and thus discourages exploration. Using Model6 does separate the two agents somewhat but crucially the average values remain negative for both agents. (a) Food-Shelter: Model1. (b) Food-Shelter: Model6.

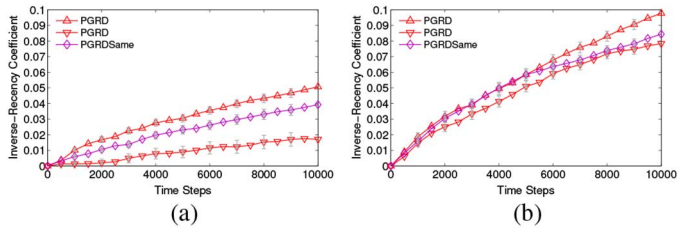


Fig. 12. Learning curves (of reward parameter) for the agents in UCT-PGRD & UCT-PGRDSame MASs in Pursuit domain using UCT-parameters $D = 3$, $T = 100$. Panel (a) is for the case when the all agents use Model1 while Panel (b) is for the case when all agents use Model6. The x-axis in each panel is time steps while the y-axis is the value of inverse-recency coefficient (see Eq. (2)). Each (learning) curve thus shows the adaptation of the inverse-recency coefficient with time steps. The results are an average over 100 independent runs, where for UCT-PGRD the agents were separated according to higher coefficient versus lower coefficient for the purposes of averaging. Using Model6 allows for faster learning of larger coefficient values to encourage exploration and this is the crucial difference relative to Model1. (a) Pursuit: Model1. (b) Pursuit: Model6.

model-type, a pair of bars are shown: the performance of UCT-ObjRe MAS and UCT-PGRD MAS for Food-Shelter, and the performance of UCT-ObjRe MAS and UCT-PGRDSame MAS for Pursuit. Several interesting observations can be made from this figure. First, it is clear that in both domains the choice of model matters to performance. Some of the models lead to significantly worse performance than other models. Indeed, for the Food-Shelter domain Models 4, 5 and 6 lead to worse performance than even Model0 (the unconditional model). Second, for each model-type learning a guidance reward function via PGRD achieves better/higher objective utility relative to using the objective reward as the guidance reward. This observation supports our basic claim that PGRD can improve performance relative to ObjRe regardless of the quality of the model used for UCT-planning, and is an important form of robustness that should make PGRD useful in practice in complex MAS environments where learning accurate models of the other agent would be difficult. Third, the differences in the two domains lead to different state variables being useful in the best performing models. Specifically, including locSelf in the models' domain significantly degrades performance in Food-Shelter (we speculate that this is because the location of the food-seeking agent varies a lot and this makes for a harder model-learning problem for the food-seeking agent). In Pursuit, there is a three-way relation among the two agents and the prey in that neither agent can independently capture the prey, and therefore the more state variables included in the model, the better the performance. For example, for both ObjRe and PGRD MASs, Model3 outperform Models 1 and 2; Model6 outperform Models 3 and 5; etc.

Finally, Fig. 10 only shows the effect of different models at the end of the lifetime of the MASs. Figures 11 and 12 show the dynamic effect of Model1 and Model6 on Food-Shelter and Pursuit respectively. As seen in Fig. 11 the crucial difference between the use of Model1 and Model6 is that Model 1 allows for specialization in which one agent learns a positive coefficient for the inverse-recency feature while the other agent learns a negative coefficient for the inverse-recency feature. As seen in Fig. 12 in Pursuit the crucial difference between the use of Model1 and Model6 is that the latter model allows for faster learning of higher positive coefficients for the inverse-recency features leading to faster learning that exploration of states not recently visited is a useful means to the end of obtaining higher objective utility.

VI. Conclusion

In this paper, we defined a new multiagent optimal rewards problem whose solution defines the guidance reward for each agent in a team such that the resulting behavior of the team ends up maximizing the objective utility of the team as measured by a given shared objective reward function. We provided a multi-agent nested optimal reward and control architecture for solving this problem and empirically demonstrated on two domains that given limited CPU-time per decision, it is better to spend some of the limited computational resources in learning good guidance reward functions instead of spending them all on planning good actions. This is the first evidence of strong-mitigation in learning guidance reward functions for multiagent settings. We also showed that our specific adaptation to multiagents of existing single-agent methods for learning guidance reward functions and for planning did reliably and stably learn good guidance reward functions. In Food-Shelter, both guidance reward function and increased behavioral specialization were seen in our results.

Finally, even though our empirical results were restricted to two-agent settings, in terms of scaling to larger number of agents we note that each agent learns a guidance reward function separately and in parallel in our architecture and thus the computational burden to any single agent of learning a guidance reward function is independent of the number of agents (even for the case of more than two agents). How our empirical findings of stable and effective learning of guidance reward functions and the resulting specialization generalize beyond our two domains awaits future theoretical and empirical analyses.

REFERENCES

[1] S. Singh, R. Lewis, A. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Trans. Autonom. Mental Develop.*, vol. 2, no. 2, pp. 70–82, Jun. 2010.

[2] J. Sorg, S. Singh, and R. Lewis, "Internal rewards mitigate agent boundedness," presented at the Int. Conf. Mach. Learn. (ICML), 2010.

[3] U. Chajewska, D. Koller, and R. Parr, "Making rational decisions using adaptive utility elicitation," presented at the Nat. Conf. Artif. Intell. (AAAI), 2000.

[4] R. Cohn, S. Singh, and E. Durfee, "Characterizing EVOI-sufficient k-response query sets in decision problems," presented at the 17th Int. Con. Artif. Intell. Statist. (AISTATS), 2014.

[5] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," presented at the Int. Conf. Mach. Learn. (ICML), 2000.

[6] A. Coates, P. Abbeel, and A. Ng, "Apprenticeship learning for helicopter control," *Commun. ACM*, vol. 52, no. 7, pp. 97–105, 2009.

[7] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robot. Autom. Mag.*, vol. 17, no. 2, pp. 55–62, Jun. 2010.

[8] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. 14th Int. Conf.: Mach. Learn.*, 1997, pp. 12–20.

[9] G. Tesauro and T. Sejnowski, "A 'neural' network that learns to play backgammon," *Neural Inf. Process. Syst. (NIPS)*, 1987.

[10] A. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," presented at the Int. Conf. Mach. Learn. (ICML), 1999.

[11] B. F. Skinner, *The Behavior of Organisms: An Experimental Analysis*. Acton, MA, USA: Copley Publishing Group, 1938.

[12] J. Schmidhuber, "Curious model-building control systems," presented at the Inte. Joint Conf. Neural Netw., 1991.

[13] J. Schmidhuber, "Artificial curiosity based on discovering novel algorithmic predictability through coevolution," presented at the IEEE Congress Evol. Comput., 1999.

[14] E. Uchibe and K. Doya, "Constrained reinforcement learning from intrinsic and extrinsic rewards," presented at the IEEE Int. Conf. Develop. Learn., London, UK, 2007.

[15] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? A typology of computational approaches," *Frontiers Neurorobot.*, 2007.

[16] P.-Y. Oudeyer, F. Kaplan, and V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 265–286, Apr. 2007.

[17] A. Baranes and P.-Y. Oudeyer, "R-iac: Robust intrinsically motivated exploration and active learning," *IEEE Trans. Autonom. Mental Develop.*, vol. 1, no. 3, pp. 155–169, Oct. 2009.

[18] S. Hart and R. Grupen, "Learning generalizable control programs," *IEEE Trans. Autonom. Mental Develop.*, vol. 3, no. 3, pp. 216–231, Sep. 2011.

[19] C. Vigorito and A. Barto, "Intrinsically motivated hierarchical skill learning in structured environments," *IEEE Trans. Autonom. Mental Develop.*, vol. 2, no. 2, pp. 132–143, Jun. 2010.

[20] S. Niekum, A. Barto, and L. Spector, "Genetic programming for reward function search," *IEEE Trans. Autonom. Mental Develop.*, vol. 2, no. 2, pp. 83–90, Jun. 2010.

[21] K. Merrick, "Intrinsic motivation and introspection in reinforcement learning," *IEEE Trans. Autonom. Mental Develop.*, vol. 4, no. 4, pp. 315–329, Dec. 2012.

[22] J. Sorg, S. Singh, and R. Lewis, "Optimal rewards versus leaf-evaluation heuristics in planning agents," presented at the 25th AAAI Conf. Artif. Intell., 2011.

[23] D. H. Wolpert and K. Tumer, "Collective intelligence," presented at the 4th Workshop Econ. Heterogeneous Interacting Agents, 1999.

[24] D. Parkes and S. Singh, "An MDP-based approach to online mechanism design," in *Proc. 17th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2003.

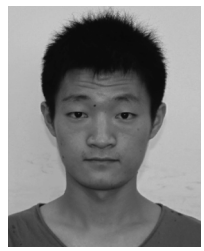
[25] P. Sequeira, F. Melo, R. Prada, and A. Paiva, "Emerging social awareness: Exploring intrinsic motivation in multiagent learning," presented at the IEEE Int. Conf. Develop. Learn., 2011.

[26] J. Bratman, S. Singh, R. Lewis, and J. Sorg, "Strong mitigation: Nesting search for good policies within search for good reward," presented at the 11th Int. Conf. Autonomous Agents Multiagent Syst. (AAMAS), 2012.

[27] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," presented at the Eur. Conf. Mach. Learn., 2006.

[28] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2–3, pp. 235–256, 2002.

[29] J. Sorg, S. Singh, and R. Lewis, "Reward design via online gradient ascent," in *Adv. Neural Inf. Process. Syst. (NIPS)*, 2010.



Bingyao Liu received the B.Eng. degree in automation from Central South University, Changsha, China, in 2008. He is currently towards the Ph.D. degree at the School of Automation Science and Electrical Engineering at Beihang University, Beijing, China.

He was a Visiting Scholar in Computer Science and Engineering at University of Michigan, Ann Arbor, from 2012 to 2013.

Satinder Singh received the B.Tech degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, in 1987, and the Ph.D. degree in computer science from the University of Massachusetts, Amherst, MA, USA in 1993.

He is a Professor of Computer Science and Engineering at the University of Michigan, Ann Arbor, MI, USA. He is Director of the Artificial Intelligence Laboratory at the University of Michigan.

Richard L. Lewis received the B.S. degree (Hons) in computer science from the University of Central Florida, Orlando, FL, USA, in 1987, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA in 1993.

He is a Professor of Psychology and Linguistics at the University of Michigan, Ann Arbor, MI, USA. He is Director of the Language and Cognitive Architecture Laboratory, and Chair of the Cognition and Cognitive Neuroscience Program at the University of Michigan.

Shiyin Qin received the bachelor's and master's degrees in engineering science in automatic controls and industrial systems engineering from Lanzhou Jiaotong University, Lanzhou, China, in 1978 and 1984, respectively, and his the Ph.D. degree in industrial control engineering and intelligent automation from Zhejiang University, Zhejiang, China, in 1990.

He is a Professor at the School of Automation Science and Electrical Engineering in Beihang University, Beijing, China.