Local Approximation of Difference Evaluation Functions

Mitchell Colby
Oregon State University
Corvallis, OR

Theodore Duchow-Pressley
Oregon State University
Corvallis, OR

Jen Jen Chung
Oregon State University
Corvallis, OR

Kagan Tumer
Oregon State University
Corvallis, OR

ABSTRACT

Difference evaluation functions have resulted in excellent multiagent behavior in many domains, including air traffic and mobile robot control. However, calculating difference evaluation functions requires determining the value of a counterfactual system objective function, which is often difficult when the system objective function is unknown or global state and action information is unavailable. In this work, we demonstrate that a local estimate of the system evaluation function may be used to estimate difference evaluations using readily available information, allowing for difference evaluations to be computed in multiagent systems where the mathematical form of the objective function is not known. This approximation technique is tested in two domains, and we demonstrate that approximating difference evaluation functions results in better performance and faster learning than when using global evaluation functions. Finally, we demonstrate the effectiveness of the learned policies on a set of Pioneer P3-DX robots.

Keywords

Multiagent reinforcement learning, difference rewards

1. INTRODUCTION

Difference evaluation functions have been shown to significantly improve learning in multiagent systems, both in the speed of convergence and the quality of the converged policy. They have been successful in many different cooperative multiagent domains, including air traffic control, network routing, multiple mobile robot control, and congestion games such as the bar problem [2, 9]. These evaluation functions have two key theoretical advantages. First, they are aligned with the overall system objective [2]. This means an individual agent acting to increase the value of the difference evaluation function also acts to improve the value of the overall system evaluation function. Second, they remove much of the noise in the feedback signal associated with other agents, allowing for individual agents to more easily learn the effects of their actions on system performance [2].

Difference evaluation functions are the *difference* between the system objective function and the *counterfactual*, or the value of the system objective function independent from the agent being evaluated. This gives an approximation of the value that the agent added to the overall system. While difference evaluation functions are conceptually simple, they are often difficult to compute. More specifically, it is difficult to calculate the counterfactual, or the value of the system objective function independent of a particular agent.

There are two key reasons counterfactuals are difficult to compute. First, calculating the counterfactual requires global knowledge of the system state and the actions taken by each agent; in practice, agents in multiagent systems rarely have such knowledge. This makes local computation of difference evaluation functions difficult in practice. Second, the system objective function itself is typically unavailable to the agents, meaning that even if agents have global knowledge, a direct computation of difference evaluations is typically unavailable. One approach to deal with calculating difference evaluations has been to leverage domain knowledge to estimate the system evaluation function.

Recent work has focused on addressing the calculation of the counterfactual in which the system evaluation function is unknown [20]. The counterfactual is estimated using a function approximator that is trained on previous values of the system evaluation function. This work extended difference evaluation functions to domains in which the analytical form of the system evaluation function is unavailable, and computing difference evaluations requires resimulation or approximation. However, this technique required that the designer had expert system knowledge as well as global state information in order to create the approximation. These are significant limitations, as the form of the system evaluation function is often unknown, and global knowledge about the system is often unavailable. In order for difference evaluations to be applicable in generic multiagent domains, we must use approximation techniques which do not require domain knowledge or global knowledge of the system state.

In this work, we present a generic and model-free approach to approximate difference evaluations. Each agent maintains a local approximation of the system evaluation functions, based on their local state and action information and the value of the system evaluation function. We assume that the only information available to agents includes their local state, their action selection, and the instantaneous value of the system evaluation function. This information is exactly what is available to agents learning with the global evaluation function, so our approximation technique does not require more information than is typically available to learning agents. The only global information needed in this approach

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

is the value of the system evaluation function, which we assume can be broadcast to each agent. Each agent uses its private function approximator to estimate the value of the difference evaluation, in order to provide a feedback signal based only on local information.

The contributions of this work are to:

- Develop an approach to approximate difference evaluation functions using information available to each agent in a multiagent system.
- Demonstrate that the proposed approximation algorithm results in up to 98% of the performance of directly computed difference evaluation functions.
- Demonstrate the effectiveness of policies learned using the proposed approximation algorithm using a set of Pioneer P3-DX robots.

In Section 2 we briefly discuss background and related work. Section 3 describes the approximation technique. Section 4 presents the two different domains in which our reward approximation method is applied. Section 5 shows the performance of our algorithm in each domain. Section 6 presents an analysis of learned policies on physical robots. Finally, in Section 7 we discuss and conclude the work.

2. RELATED WORK

The following sections introduce difference evaluation functions, as well as related work involving approximating evaluation functions to provide agent feedback.

2.1 Difference Evaluation Functions

The agent-specific *difference evaluation function* $D_i(s_i, a_i)$ is defined as [2]:

$$D_i(s_i, a_i) = G(s, a) - G(s_{-i} \cup c_{s,i}, a_{-i} \cup c_{a,i}) \tag{1}$$

where:

- $G(s, a)$ is the total system evaluation
- s is the joint system state
- a is the system joint-action
- s_{-i} are all the states on which agent i has no effect
- a_{-i} are all the agent actions not involving agent i,
- $c_{s,i}$ and $c_{a,i}$ are fixed states and actions not dependent on agent i.

Intuitively, the second term (the *counterfactual*) in Equation 1 evaluates the system without the effects of agent i, so the difference evaluation gives agent i's contribution to the system evaluation [2]. Note that:

$$\frac{\partial D_i(s, a)}{\partial a_i} = \frac{\partial G(s, a)}{\partial a_i} \tag{2}$$

where a_i is the action taken by agent i. Thus, an agent acting to increase the value of its difference evaluation function will also act to increase the value of the overall system evaluation function. This property is termed *factoredness* [2]. Also note that $D_i(s, a)$ removes all elements of the system evaluation function not affected by agent i, decreasing noise and providing agent-specific feedback which is strongly coupled to a particular agent's actions. This property is termed *learnability* [2]. In addition to the theoretical properties of factoredness and learnability, difference evaluations have been shown to improve the probability of agents finding best response actions in cases where tight coordination is required to reach optimal Nash equilibria [10].

Although factoredness and learnability allow for difference evaluations to provide very useful agent feedback during learning, there are two key limitations to these functions. First, calculating the counterfactual in difference evaluations requires global knowledge about the system state and joint action. In practice, such information is rarely available. Therefore, calculating $D_i(s, a)$ analytically requires a centralized mechanism to provide agent feedback, which is generally unavailable. Second, the system evaluation function is typically unknown to agents, making direct calculation of the counterfactual term unfeasible in some cases. In order to extend difference evaluations to general multiagent systems, techniques for approximating the system evaluation function using only local information are required.

2.2 Approximation of Evaluation Functions

In cases where the system evaluation function is unknown, it may be approximated in order to provide better feedback to learning agents [3, 5]. In the case of reinforcement learning, reward functions are modeled; in the case of evolutionary algorithms, fitness functions are modeled. In either case, approximation of system evaluation functions allows for agent evaluation functions to be easily shaped.

Function approximation is commonly seen in reinforcement learning applications, where the value function or system state is approximated due to limited state information [11, 12, 16]. Value functions are often approximated when only partial state information is available, often using neural networks, maps, or some other type of function approximator [1, 6, 18]. The conclusion that only partial state information is necessary to accurately model value functions suggests these types of approaches can be successfully extended to multiagent learning problems, where partial state information is available to each agent.

Fitness functions have also been approximated for use in evolutionary algorithms [7, 15]. This fitness approximation is typically used because the number of fitness function evaluations dominates the optimization cost in evolutionary algorithms, and fitness approximation typically decreases time to convergence [13, 17]. However, these approaches can easily be extended to cases in which the mathematical form of the fitness function is unknown, as in cases where the difference evaluation must be approximated. All of the methods described above are single agent cases, but there has also been research investigating approximating system evaluation functions in multiagent learning settings.

Difference evaluation functions have been approximated in air traffic control domains [20]. A tabulated linear function consisting of neural networks approximates the system evaluation function. Each neural network approximated a specific aspect of the air traffic domain (congestion, delay), and a weighted sum of the network outputs was used to provide the approximation of the overall system evaluation function. This approach yielded accurate approximations of the system evaluation function and allowed for difference evaluation functions to be accurately estimated to provide agent-specific feedback during learning.

There are two key drawbacks to this approach. First, expert domain knowledge was needed to create the approximation of the system evaluation function, and such knowledge is not always available. Second, this approach required global knowledge of the system state, which is often unavailable in a distributed multiagent system. Although this approach gave good results for approximating difference evaluation functions in the air traffic domain, it does not extend to any generic multiagent domain.

There is a wide range of research involving approximation in multiagent learning. However, very little research has been conducted on approximating difference evaluations. More importantly, the research which has been conducted on approximating difference evaluations has involved approximations which were dependent on expert system knowledge as well as global state and action information. In order for difference evaluations to be approximated in generic multiagent settings, the requirements for expert domain knowledge must be eliminated, and the approximations must be constructed using only local information.

3. DIFFERENCE EVALUATION APPROXIMATION

In this section, we introduce a general algorithm for approximating difference evaluations, and then demonstrate how this algorithm is implemented in different types of domains. This algorithm was first presented in [8] for continuous state and action domains, and we now present the generalized approximation algorithm and explain how it can be implemented in different types of domains. In general, multiagent systems may be stateless or stateful, and may have continuous or discrete state and action spaces. We select two types of domains to demonstrate implementation of our algorithm: a stateless, discrete action domain as well as a continuous state and action domain. For the purpose of this analysis, we assume a cooperative multiagent system aims to maximize some system objective function $G(s, a)$, and that the value of $G(s, a)$ is available to each agent. Recall that the difference evaluation is defined as:

$$D_i(s, a) = G(s, a) - G(s_{-i} \cup c_{s,i}, a_{-i} \cup c_{a,i}).$$

Since we assume that the value of $G(s, a)$ is available to each agent, approximating the difference evaluation function only requires approximating the $G(s_{-i} \cup c_{s,i}, a_{-i} \cup c_{a,i})$ term. Given the particular domain, a suitable function approximator is chosen. For a stateless domain with discrete actions, this approximator may simply be a vector. For a stateful domain with continuous actions, this approximator may be a neural network. Note that these are not the only possible options for these function approximators, but are potential choices that match the domains they will be used in.

The difference evaluation approximation algorithm is presented in Algorithm 1. At each time step, each agent takes an action, and $G(s, a)$ is computed and broadcast to each agent. Each agent i maintains a private approximation for $G(s, a)$, denoted as $\hat{G}_i(s_i, a_i)$. The only information available to the agent includes the agent's state, action choice, and the broadcast value of $G(s, a)$. Each agent's approximation of $G(s, a)$ is a mapping from that agent's state and action to the system objective function. This approximation is initially very noisy, because the state and action information used to approximate $G(s, a)$ is limited only to one agent's state and action. However, as learning progresses, the policies of each agent begin to converge; as the policies of other agents converge, approximating $G(s, a)$ using only one agent's state and action becomes less noisy, because the variance in the joint action for a particular system-level state is reduced. Given an agent's approximation $\hat{G}_i(s, a)$, the difference evaluation function is estimated as:

$$\hat{D}_i(s, a) = G(s, a) - \hat{G}_i(c_{s,i}, c_{a,i}) \tag{3}$$

where $\hat{D}_i(s, a)$ is agent i's approximation of the difference evaluation function. In order to evaluate $\hat{G}_i(c_{s,i}, c_{a,i})$, a default state and default action are chosen for each agent. In other words, the approximation of the difference evaluation function determines *the difference between the system objective function and the approximated system objective function if agent i took a default action.*

```
Initialize $N$ agents
foreach Agent do
    Initialize private function approximator $\hat{G}_i(s_i, a_i)$
end
foreach Learning Step do
    foreach Agent do
        collect local state information $s_i$
        select action $a_i$ using agent's policy
    end
    Calculate $G(s, a)$ based on joint action
    Broadcast $G(s, a)$ to each agent
    foreach Agent do
        Update $\hat{G}_i(s_i, a_i)$ using $s_i$, $a_i$, and $G(s, a)$
        $\hat{D}_i(s, a) = G(s, a) - \hat{G}_i(c_{s,i}, c_{a,i})$
        Update policy/value table based on $\hat{D}_i(s, a)$
    end
end
```

Algorithm 1: Approximation of difference evaluation functions. The functional form of $\hat{G}$ depends on the specific domain. For example, a stateless discrete action domain may use a vector, while a continuous state and action domain may use a neural network.

The implementation of this approximation approach in a stateless discrete action domain as well as a continuous state and action domain are detailed in the following sections. We demonstrate how a multiagent reinforcement learning algorithm with an approximation of difference evaluations may be implemented in the stateless discrete action domain, and how a cooperative coevolutionary algorithm with an approximation of difference evaluations may be implemented in a continuous state and action domain. Difference evaluations are commonly used in both reinforcement learning and cooperative coevolutionary algorithms, so we choose to demonstrate how both types of algorithms may incorporate approximate difference evaluations.

3.1 Stateless Discrete Action Domains

We now demonstrate how Algorithm 1 may be implemented for a multiagent reinforcement learning algorithm in a stateless discrete action domain. The implementation of Algorithm 1 in a multiagent reinforcement learning problem with a stateless discrete action domain is given in Algorithm 2. Note that such problems may also be solved with coevolutionary algorithms. In a stateless discrete action domain, each agent maintains a value vector $V_i(a)$ containing the expected value of each possible action. Each agent also maintains an approximation of $G(a)$, which is simply a vector containing the estimated value of the system evaluation function corresponding to each action the agent may take. At each learning step, each agent i selects an action a_i. The system evaluation function $G(a)$ is then calculated, and this value is broadcast to each agent in the system. Each agent

```
Initialize $N$ agents
foreach Agent do
    Initialize private value table $V_i(a)$
    Initialize private function approximator $\hat{G}_i(a)$
end
foreach Learning Step do
    foreach Agent do
        select action $a_i$ using agent's policy
    end
    Calculate $G(s,a)$ based on joint action
    Broadcast $G(s,a)$ to each agent
    foreach Agent do
        $\hat{G}_i(a_i) \leftarrow \alpha_1 \cdot \hat{G}_i(a_i) + (1-\alpha_1) \cdot G(a)$
        $\hat{D}_i(a_i) = G(a) - \hat{G}_i(a_0)$
        $V_i(a_i) \leftarrow \alpha_2 \cdot V_i(a_i) + (1-\alpha_2) \cdot \hat{D}_i(a)$
    end
end
```

Algorithm 2: Stateless Discrete-Action Multiagent Reinforcement Learning using $D_i(s,a)$ Approximation

then updates its approximation $\hat{G}_i(a_i)$ according to:

$$\hat{G}_i(a_i) \leftarrow (1-\alpha_1) \cdot \hat{G}_i(a_i) + \alpha_1 \cdot G(a) \tag{4}$$

Once each agent has updated its approximation of $G(a)$, the difference evaluation for each agent is estimated as:

$$\hat{D}_i(a) = G(a) - \hat{G}_i(c_{a,i}) \tag{5}$$

where $c_{a,i}$ is some default action. The value table is updated using the approximate difference evaluation function:

$$Q_i(a_i) \leftarrow (1-\alpha_2) \cdot Q_i(a_i) + \alpha_2 \cdot \hat{D}_i(a) \tag{6}$$

3.2 Continuous State and Action Domains

We now demonstrate how Algorithm 1 may be implemented in a cooperative coevolutionary algorithm in a continuous state and action domain. We assume each agent has a neural network policy that maps the agent's state to an action. Further, each agent has a two-layer feedforward neural network approximation of $G(s,a)$. Note that neural networks are not the only form of function approximation that may be used, but are chosen to demonstrate this example. At each timestep, each agent keeps track of its state and chosen actions. The value of the system evaluation function $G(s,a)$ is broadcast to each agent after each agent has taken an action. Then, each agent updates its approximation $\hat{G}_i(s_i,a_i)$ using the $(s,a,G(s,a))$ tuple and backpropagation, where the error in the approximation is $G(s,a) - \hat{G}_i(s_i,a_i)$.

The difference evaluation is then approximated using Equation 3, and the estimated value of $\hat{D}_i(s,a)$ is used to assign the fitness associated with the state-action pairing selected by the agent's policy. Agents are selected for survival using ϵ-greedy selection. The implementation of Algorithm 1 using a cooperative coevolutionary algorithm is given in Algorithm 3.

```
Initialize $N$ populations of $k$ neural networks
foreach Population $i$ do
    foreach Individual $j$ do
        Initialize function approximator $\hat{G}_{i,j}(s_i,a_i)$
    end
end
foreach Generation do
    foreach Population do
        produce $k$ successor solutions
        mutate successor solutions
    end
    for $i = 1 \rightarrow 2k$ do
        randomly select one agent from each population
        add agents to team $T_i$
        foreach Simulation Timestep do
            each agent $j$ in team $T_i$ finds local state $s_i$
            each agent $j$ in team $T_i$ chooses action $a_i$
            $G(s,a)$ is broadcast to each agent
            update $\hat{G}_{i,j}(s_i,a_i)$ based on $s_i$, $a_i$, $G(s,a)$
            $\hat{D}_{i,j}(s,a) = G(s,a) - \hat{G}_{i,j}(c_{s,i},c_{s,a})$
            each agent $j$ increments fitness by $\hat{D}_{i,j}(s,a)$
        end
    end
    foreach Population do
        select $k$ networks using $\epsilon$-greedy
    end
end
```

Algorithm 3: Continuous State and Action CCEA using $D_i(s,a)$ Approximation

4. EXPERIMENTAL DOMAINS

In this section, we introduce the two test domains utilized in this work, and provide a detailed description of the system dynamics and evaluation functions used in each domain. The first domain, the El Farol bar problem, is a stateless congestion domain with a discrete action space. The second domain, the rover domain, is a stateful domain with continuous states and actions. These domains were chosen to highlight that our approximation approach can be applied in a wide range of different multiagent problems.

4.1 Bar Problem

The El Farol Bar Problem [4] is a frequently-used abstraction of congestion problems. We use a multi-night modification of the El Farol Bar Problem. In this problem there is a capacity c that provides the most enjoyment for everyone who attends the bar on that particular night. This is a stateless one shot problem where agents choose the night they attend the bar, and receive a reward based on their enjoyment. The traditional bar problem local reward is a function of the attendance of that night:

$$L_i = e^{\frac{-x_i(a_i)}{c}} \tag{7}$$

where $x_i(a_i)$ is the attendance on the night agent i went to the bar. The system-level reward is a simple summation of these local rewards across all agents:

$$G(a) = \sum_{k=0}^{K} x_k(a) e^{\frac{-x_k(a)}{c}} \tag{8}$$

where k is the index of the night, and $x_k(a)$ is the number of agents who attended on the kth night.

4.2 Rover Domain

In the rover domain [2, 9], a set of rovers on a two dimensional plane must coordinate in order to observe points

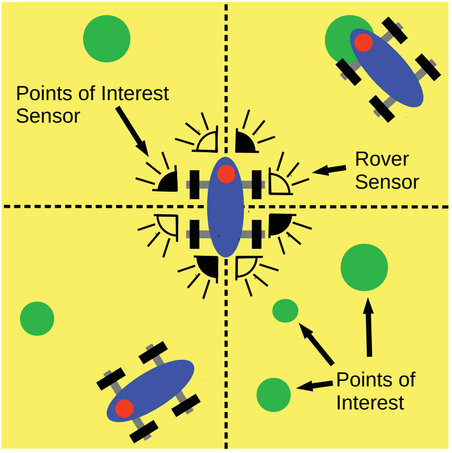


Figure 1: Rover Domain Representation. Each rover senses the POIs and other rovers within an observation radius in each of its four sensing quadrants.

of interest (POIs) scattered across the domain (Figure 1). Each POI has an associated value, and each observation of a POI made by a rover yields an observation value which is inversely proportional to the distance that the rover is from the POI. The distance metric used in this domain is the squared Euclidian norm, bounded by a minimum observation value to prevent division by zero:

$$\delta(x, y) = min \left\{ ||x - y||^2, \delta^2_{min} \right\} \tag{9}$$

The objective of the rovers is to maximize the observation values of the POIs over an episode, given by:

$$G = \sum_t \sum_j \frac{V_j}{min_i \delta(L_j, L_{i,t})} \tag{10}$$

where V_j is the value of POI j, L_j is the location of POI j, and $L_{i,t}$ is the location of the ith rover at time t.

Although any rover may observe any POI, the system evaluation only takes into account the closest observation made for each POI. The POI locations are static throughout each experiment. The rovers have eight total sensors, two sensors per quadrant. The first sensor for quadrant q returns the sum of the values of POIs divided by their squared distance to the rover:

$$s_{1,q,i} = \sum_{j \in I_q} \frac{V_j}{\delta(L_j, L_i)} \tag{11}$$

where I_q is the set of POIs in quadrant q. The second sensor returns the sum of square distances from a rover to all the other rovers in the quadrant:

$$s_{2,q,i} = \sum_{i' \in N_q} \frac{1}{\delta(L_{i'}, L_i)} \tag{12}$$

where N_q is the set of rovers in quadrant q. The eight sensors give an approximate representation of the world, reducing the location and number of rovers and POIs in each quadrant to an average number. This representation of the rover domain is used as a low-fidelity model of a set of differential drive robots. Section 6 discusses how policies found in this domain can be mapped to a physical robotic system.

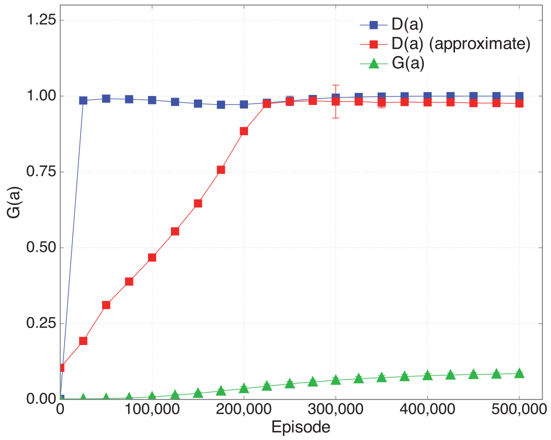


Figure 2: Bar problem results. When approximating $D_i(a)$, the learning rate is slower than when using $D_i(a)$, but the converged performance is nearly identical. $D_i(a)$ performs almost 10 times better than the overall system evaluation function $G(a)$.

5. EXPERIMENTAL RESULTS

The following sections detail the experimental results in the bar problem and the rover domain. One bar problem experiment was conducted, with 1000 agents in the system. Two rover domain experiments were conducted, varying between 10 and 100 agents to determine the effects of system size on the performance of our approximation algorithm.

5.1 Bar Problem Domain Results

The bar problem was initialized as follows. There are 1000 agents and 10 nights, where each night has a capacity of 10. The learning rate α_1 for the approximation of the system evaluation function was set to 0.1. The learning rate α_2 for the value table was set to 0.1. Each experimental run lasted for $500,000$ timesteps, and 100 statistical runs were conducted. The experimental results are shown in Figure 2, and the reported error bars are error in the mean σ/N^2.

As seen in Figure 2, approximating the difference evaluation function results in almost 10 times better performance than using the system evaluation function $G(a)$. When approximating $D_i(a)$, the solution takes much longer to converge than when analytically computing $D_i(a)$. However, converged performance of actual and estimated difference evaluation functions is nearly identical.

It is of note that although the analytical calculation of $D_i(a)$ results in faster learning, it is often impossible to analytically compute the system evaluation function in multiagent learning systems. Often, the mathematical form of $G(a)$ is unknown, meaning $D_i(a)$ cannot be directly computed. However, agents can still use local knowledge to approximate $G(a)$ and thus estimate difference evaluation functions, which drastically improve system performance. So, even though the analytical computation of $D_i(a)$ provides faster learning than when approximating $D_i(a)$, it is not always possible to perform this analytical computation.

The key result is that agents with only local knowledge converge to the same performance (although in more computational time) as agents with global knowledge about the system. In cases where global knowledge is available, it is often beneficial to use this knowledge while shaping agent feedback signals. However, in many cases, agents' knowledge is often limited to what they can observe, and constructing meaningful agent feedback based on this limited information is critical for ensuring high system performance. These results demonstrate that in some cases, approximate difference evaluations result in no significant loss in converged system performance when only local knowledge is available.

5.2 Rover Domain Results

The rover domain was initialized as follows. There are 10 agents and 10 POIs in a planar world of size 25 units by 25 units. For coevolution, each agent maintains a population of 25 neural networks. Networks are mutated by adding variables drawn from a Gaussian distribution with zero mean and unit variance to 10 randomly drawn weights from each neural network. Each episode lasts 25 timesteps, and agents can move a maximum of 1 distance unit per timestep. At the beginning of each episode, agent positions are randomly initialized near the center of the domain. The coeovlutionary algorithm[1] was run for 3000 generations, and 150 statistical runs were conducted for statistical significance. The experimental results are shown in Figure 3. As in the bar problem, reported error bars represent the error in the mean of the statistical data.

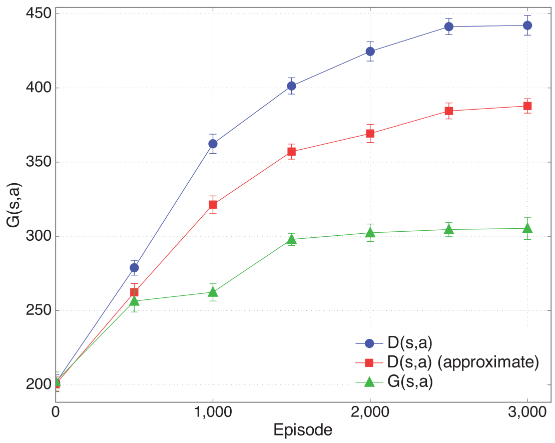


Figure 3: Rover domain results (10 agents). Approximating $D_i(s,a)$ results in 88% of the performance attained when analytically computing $D_i(s,a)$. Approximating the difference evaluation function results in significant performance gains when compared to using the system evaluation function $G(s,a)$.

As seen in Figure 3, approximating $D_i(s,a)$ results in an approximate 23% increase in converged performance compared to using the system evaluation function $G(s,a)$. When approximating $D_i(s,a)$, the converged performance was 88% of the converged performance when analytically computing $D_i(s,a)$. Although the performance decreases when approximating rather than analytically computing $D_i(s,a)$, it is important to note that much less information is required to make the approximation. The approximation of $D_i(s,a)$ requires only an agent's local state and action, and the evaluation of $G(s,a)$. In order to analytically compute $D_i(s,a)$, global information about the state and action of each agent is required, as well as the functional form of $G(s,a)$. So, although estimating $D_i(s,a)$ results in slightly worse performance than analytically computing $D_i(s,a)$, it is often extremely difficult to make this analytical calculation, meaning that approximation is required in order to implement difference evaluation functions.

Unlike the bar problem, there is a drop in converged performance between $D(s,a)$ and $\hat{D}_i(s,a)$. This can be attributed to the increased complexity of the state and action space in the rover domain. In the stateless bar problem, there were simply 10 discrete actions that agents had to choose from. In the rover domain, both the states and actions are continuous. In the bar problem, difference evaluations attempt to answer whether an agent had a positive impact on the system for the specific night the agent attended; in the rover domain, difference evaluations attempt to answer whether the agent had a positive impact on information collection over the course of a time-extended task. Clearly, approximating the system evaluation function in the rover domain is much more complex than in the bar problem, and explains why the approximation resulted in decreased performance of $\hat{D}_i(s,a)$ when compared to $D(s,a)$.

It is important to note that a local approximation only resulted in a 12% decrease in performance when the state information used to construct agent feedback decreased by 90%. The states and actions of all 10 agents are incorporated into calculating $D(s,a)$. In the approximation $\hat{D}_i(s,a)$, only one agent's state is used in the calculation. This is an extremely promising result, as a massive loss in information results in only a small loss in performance.

Another important note is that when learning initially begins, $\hat{D}_i(s,a)$ is simply a noisy version of $\hat{G}(s,a)$. As learning progresses and agent approximations of $G(s,a)$ become more accurate, $\hat{D}(s,a)$ gradually moves from a noisy representation of $\hat{G}(s,a)$ to an approximation of $D_i(s,a)$. This is why early in learning, $\hat{D}_i(s,a)$ performs closer to $G(s,a)$, but gradually performs closer to $D_i(s,a)$.

5.3 Scaling Results

In order to demonstrate the scalability of our approach, we use the difference evaluation approximation algorithm in a larger instance of the rover domain. In this experiment, there are 100 agents and 100 POIs in a planar world of size 50 by 50. Learning is allowed to proceed for 5000 generations. All other parameters of the experiment are identical to the rover experiment in Section 5.2. The experimental results are shown in Figure 4. The reported error bars represent the error in the mean of the statistical data.

As seen in Figure 4, learning with $\hat{D}_i(s,a)$ results in 79% of the performance compared to using the analytically computed difference evaluation, and outperforms the system evaluation function by 49%. Although $\hat{D}_i(s,a)$ performed more poorly compared to $D_i(s,a)$ in this larger domain (79% vs. 88%), it outperformed $G(s,a)$ by a wider margin (49% vs. 23%) as the system grew in complexity. This is strong evidence supporting the use of approximate difference evaluations over the system evaluation function.

[1] We also performed these experiments using multiagent Q-learning using policy gradient methods. The cooperative coevolutionary algorithm results are presented to demonstrate that our approximation of difference evaluations applies to both multiagent reinforcement learning and cooperative coevolutionary algorithms, not to suggest that coevolutionary algorithms are the preferred approach for the rover domain.

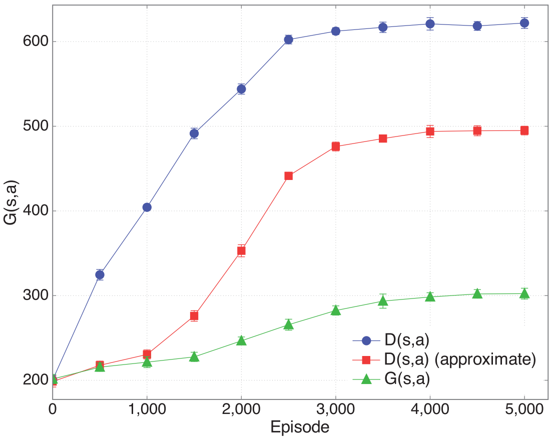


Figure 4: Rover domain results (100 agents). Approximating $D_i(s,a)$ results in 79% of the performance attained when analytically computing $D_i(s,a)$. Approximating the difference evaluation function results in significant performance gains when compared to using the system evaluation function $G(s,a)$.

It is of note that in this larger domain, learning occurs more slowly (as compared to $D_i(s,a)$) than in the 10 agent domain. This can be attributed to the fact that the approximation of $G(s,a)$ is more difficult to learn in the larger domain, and poor approximations early on in learning decrease the learning speed. As the approximator becomes more accurate, learning with $\hat{D}_i(s,a)$ speeds up. However, even when agents use only local knowledge and approximate $D_i(s,a)$, they still significantly outperform agents using $G(s,a)$. These results indicate that difference evaluations may be extended to applications where global knowledge about the system state, as well as the form of the system evaluation function, are unavailable.

It is of note that in all three experiments conducted, approximating difference evaluations instead of directly calculating them required at least 90% less system information, but only resulted in performance losses of up to 21%. This is an extremely promising result, as using far less knowledge to calculate agent feedback results in a comparatively low drop in system performance.

5.4 Theoretical Implications of $D(z)$ Approximation

The difference evaluation is fully factored, meaning that an agent acting to increase the value of $D_i(s,a)$ will always also increase the value of $G(s,a)$. However, when approximating the difference evaluation, the guarantee of factoredness is lost. We now analyze the degree to which the approximate difference evaluation is factored with respect to the system evaluation function.

In order to evaluate the degree of factoredness associated with the approximations, we performed random sampling in the rover domain to compare $G(s,a)$ with $\hat{G}_i(s_i,a_i)$. Random states are created by drawing the positions of the rovers and POIs from a uniform random distribution such that rovers and POIs could be placed at any location in the planar world. Then, a random action was selected (x- and $y-$motion) from uniform random distributions for a randomly selected rover. This action was executed, and the change in the value of $G(s,a)$ was recorded. If the sign of this change is equal to the sign of $\hat{G}_i(s_i,a_i)$, then $\hat{G}_i(s_i,a_i)$ is factored with $G(s,a)$ for this particular state-action pair. This process was repeated for one million randomly generated states.

We found that $\hat{G}_i(s_i,a_i)$ was factored with $G(s,a)$ in 94% of the states tested in the 10 agent domain, and 78% of the states tested in the 100 agent domain. This is a particularly interesting result for two reasons. First, the approximation was capable of attaining a fairly high degree of factoredness, which explains why the approximation of difference evaluations provided good performance. Second, when the approximation of $G(s,a)$ was found to be 95% factored in the 10 agent domain, the approximation of difference evaluations led to 88% of the performance of analytically calculated difference evaluations. When the approximation of difference evaluations was 78% factored, the approximation of difference evaluations led to 79% of the performance of analytically computed difference evaluations. In our experiments, the degree of factoredness of $\hat{G}_i(s_i,a_i)$ is strongly correlated with the performance of $\hat{D}_i(s,a)$.

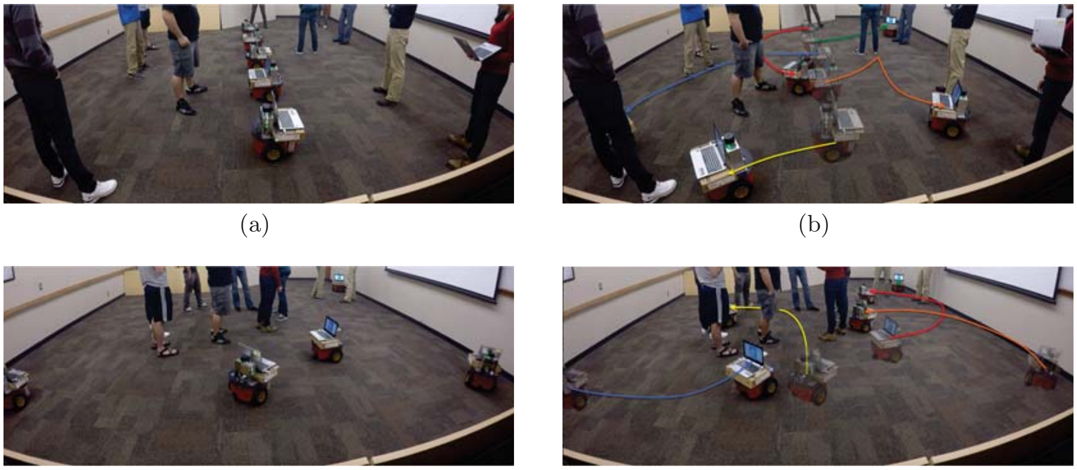
6. HARDWARE IMPLEMENTATION

We implemented the rover domain on a set of five Pioneer P3-DX compact differential-drive research robots. Each robot was equipped with odometry and lidar sensors, and so could determine its own position and the positions of surrounding objects. Each robot was controlled by a computer running the Robot Operating System (ROS) [21]. POIs were defined as people standing with the robots (Figure 5).

We mapped the rover domain as described in Section 4 onto the hardware implementation by reducing the variety of information brought in by the robots' sensors down to the 8 state variables described above. To mimic the rover sensor return for each quadrant, each robot reported its position to a central hub, using the `rocon` multimaster system for ROS [22]. The hub calculated the relative robot positions and sent each robot its four quadrant sensor values. This centralized calculation was necessary since the lidars were mounted above the robot platform such that their planar laser scans could not reliably detect the small available surface area of the other robots. Future work will investigate improved designs for robust robot detection. The POI sensor values were extracted by feeding the raw lidar returns into the trained people classifier of the `leg_detector` ROS package developed by Willow Garage [19]. The `leg_detector` package output the target position of human POIs in the robot body frame, and this data was condensed into the 4 POI quadrant values using Equation 11.

Rover control policies were trained in a 5-rover, 10-POI simulation (5000 generations) and used the 8 state input variables (2 sensor values for each quadrant, as discussed above) to output a displacement command $(\Delta x, \Delta y)$ in the respective body frame. These values were issued as waypoint commands to the robot through the `move_base` ROS package [14]. This allowed the robot to use the built-in obstacle-avoidance routines to find collision-free paths to the locations dictated by the control policy.

Figure 5 shows the initial and final robot configurations for two sequential trials, as well as the paths taken by each robot. As seen in Figure 6, 9 POIs were successfully ob-



(a)

(b)

(c)

(d)

Figure 5: The initial configuration of the robots and POIs is shown in (a), after $58\,\mathrm{s}$ 9 of the 10 POIs have been observed and the trajectories are plotted in (b). At $59\,\mathrm{s}$, the POIs reconfigured to form two clusters, a large cluster of 8 in the center and a small cluster of 2 towards the far right as shown in (c). The control policies on the robots allowed them to reobserve all the POIs. Note that the majority of the lead time was due to the internal path planners on the robots, shown in (d). Also of note is that the robot in the far right corner failed to find a safe motion path, however the team of robots was still able to reacquire all POIs.

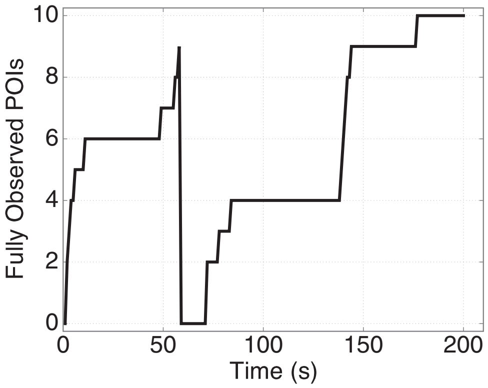


Figure 6: Hardware results from Pioneer P3-DX experiments, with 5 robots and 10 POIs. In the first POI configuration, 9 POIs were fully observed. In the second configuration, all 10 POIs were fully observed.

served in the first trial, and all 10 POIs were successfully observed in the second trial. These results are consistent with the simulation results shown in Section 5.

7. DISCUSSION

Difference evaluations have been empirically shown to improve coordination in multiagent systems in a many domains, including air traffic control, rover control, sensor network control, and congestion games. Further, difference evaluations are fully factored, and are typically low in noise. However, a key limitation of difference evaluations is the requirement for global knowledge about the state of the system as well as the system evaluation function. Thus, directly implementing difference rewards in generic multiagent domains is often a difficult task.

In this work, we demonstrate that difference evaluations may be approximated by each agent using only local state and action information. The only assumption is that the value of the system evaluation function $G(s,a)$ can be broadcast to each agent, which in most cases is a reasonable assumption. We present an approach for approximating difference evaluation functions in order to provide agent-specific feedback to improve coordination, and demonstrate in two domains the effectiveness and scalability of our approach.

The key contribution of this work is presenting a novel method to implement difference evaluation functions in any generic multiagent system, without requiring global knowledge about the state of the system or the mathematical form of the system evaluation function. Further, this approximation approach significantly outperforms methods which use the overall system evaluation function. As each agent maintains a local approximation of the system evaluation function, increases in computational cost are insignificant, because the computation is parallelized across each agent in the system.

An analysis of the relationship between the degree of factoredness of the function approximation and the performance relative to difference evaluations is the most interesting future research topic. In the 10 agent domain, the approximation of $G(s,a)$ was 94% factored and resulted in 88% of the performance of $D_i(s,a)$ without approximation. In the 100 agent domain, the approximation of $G(s,a)$ was 78% factored and resulted in 79% of the performance of $D_i(s,a)$ without approximation; this is an intriguing result, although without further research no claims can be made. There is a clear relationship between the factoredness of the approximation and the resultant system performance, but the nature of this relationship should be further investigated.

Acknowledgements

This work was partially supported under NASA NNX14AI10G.

REFERENCES

[1] J. Abounadi, D. Bertsekas, and V. Borkar. Stochastic Approximation for Non-Expansive Maps: Application to Q-Learning Algorithms. Technical report, SIAM Journal on Control and Optimization, 1998.

[2] A. Agogino and K. Tumer. Analyzing and Visualizing Multiagent Rewards in Dynamic and Stochastic Environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[3] K. Anderson and Y. Hsu. Genetic crossover strategy using an approximation concept. In *In IEEE Congress on Evolutionary Computation*, pages 527–533, 1999.

[4] W. B. Arthur. Inductive Reasoning and Bounded Rationality (The El Farol Problem). In *Amer. Econ. Review 1994*, volume 84, pages 406–411, 1994.

[5] J. Branke, C. Schmidt, and H. Schmeck. Efficient Fitness Estimation in Noisy Environment. In *Proceedings of Genetic and Evolutionary Computation*, pages 243–250, 2001.

[6] D. Brillinger. Learning a Potential Function From a Trajectory. *Signal Processing Letters, IEEE*, 14(11):867–870, 2007.

[7] D. Buche, N. Schraudolph, and P. Koumoutsakos. Accelerating Evolutionary Algorithms Using Fitness Function Models. In *Proc. Workshops Genetic and Evolutionary Computation Conference*, 2003.

[8] M. Colby, J. J. Chung, and K. Tumer. Implicit adaptive multi-robot coordination in dynamic environments. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[9] M. Colby and K. Tumer. Shaping Fitness Functions for Coevolving Cooperative Multiagent Systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Valencia, Spain, June 2012.

[10] M. Colby and K. Tumer. An evolutionary game theoretic analysis of difference evaluation functions. In *In Proceedings of Genetic and Evolutionary Computation Conference*, 2015.

[11] C. Gaskett, L. Fletcher, and A. Zelinsky. Reinforcement Learning for Visual Servoing of a Mobile Robot. In *In Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000)*, 2000.

[12] Y. Isukapalli. An Analytically Tractable Approximation for the Gaussian Q-Function. *IEEE Communications Letters*, 12(9):669 – 671, 2008.

[13] Y. Jin. A Comprehensive Survey of Fitness Approximation in Evolutionary Computation. *Soft Comput.*, 9(1):3–12, Jan. 2005.

[14] E. Marder-Eppstein. ROS Wiki - move_base. http://wiki.ros.org/move_base. Accessed: 2015-11-13.

[15] J. Martikainen and S. Ovaska. Fitness Function Approximation by Neural Networks in the Optimization of MGP-FIR Filters. In *Adaptive and Learning Systems, 2006 IEEE Mountain Workshop on*, pages 7–12, 2006.

[16] H. Murao and S. Kitamura. Incremental State Acquisition for Q-Learning by Adaptive Gaussian Soft-Max Neural Network. In *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*, 1999.

[17] V. Oduguwa and R. Roy. Multiobjective Optimization of Rolling Rod Product Design Using Metamodeling Approach. *In Genetic and Evolutionary Computation Conference*, pages 1164–1171, 2002.

[18] P. Pandey and D. Pandey. Reduct Based Q-learning: an Introduction. In *Proceedings of the 2011 International Conference on Communication*, ICCCS '11, pages 285–288, New York, NY, USA, 2011. ACM.

[19] C. Pantofaru. ROS Wiki - leg_detector. http://wiki.ros.org/leg_detector. Accessed: 2015-11-13.

[20] S. Proper and K. Tumer. Modeling Difference Rewards for Multiagent Learning (Extended Abstract). In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, June 2012.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: An open-source Robot Operating System. In *International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.

[22] D. Stonier, J. Lee, and H. Kim. ROS Wiki - rocon. http://wiki.ros.org/rocon. Accessed: 2015-11-13.