Multi-level Fitness Critics for Cooperative Coevolution

Golden Rockefeller
rockefeg@oregonstate.edu
Oregon State University
Corvallis, Oregon

Shauharda Khadka
shauharda.khadka@intel.com
Intel Corporation
San Diego, California

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

ABSTRACT

In many multiagent domains, and particularly in tightly coupled domains, teasing an agent's contribution to the system performance based on a single episodic return is difficult. This well-known difficulty hits state-to-action mapping approaches such as neural networks trained by evolutionary algorithms particularly hard. This paper introduces fitness critics, which leverage the expected fitness to evaluate an agent's performance. This approach turns a sparse performance metric (policy evaluation) into a dense performance metric (state-action evaluation) by relating the episodic feedback to the state-action pairs experienced during the execution of that policy. In the tightly-coupled multi-rover domain (where multiple rovers have to perform a particular task simultaneously), only teams using fitness critics were able to demonstrate effective learning on tasks with tight coupling while other coevolved teams were unable to learn at all.

KEYWORDS

Fitness; Critics; Cooperative Coevolution; Multiagent Learning

ACM Reference Format:
Golden Rockefeller, Shauharda Khadka, and Kagan Tumer. 2020. Multi-level Fitness Critics for Cooperative Coevolution. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Evolutionary algorithms (EAs) have been successfully applied to sequential problems, including games [14, 17], air traffic control [2], and robot control [1, 19, 26]. In most single-agent domains, the episodic feedback an agent receives from the environment is sufficient to produce good policies using evolutionary algorithms. However, in multiagent domains, the episodic *immediate feedback* does not generally reflect the potential contribution of an individual agent. This problem worsens in *tightly coupled* multiagent systems, where agents need to take complementary actions simultaneously to achieve a task.

To improve the quality of the selective pressure that the EA applies on the agent's policies, the EA can use the individual agent's *expected feedback* (with respect to varied teams) as a policy's fitness value, rather than using the immediate feedback. Using the expected feedback prevents an agent from receiving bad feedback on a potentially good action in cases where they were paired with poor teammates. Fitness approximation and sampling-based methods are two broad methods for estimating expected feedback. Sampling-based methods, such as averaging multiple evaluations, leniency [22], or hall of fame [24] need large numbers of samples to properly evaluate policies. Fitness approximation aims to capture a functional model that relates policy parameters to the expected fitness but does not reuse the information that is learned about one policy when evaluating unrelated policies that exhibit similar behaviors. Both methods suffer from the inefficient use of prior information, particularly when sampling potentially good joint-actions is difficult.

This paper introduces *fitness critics*, which are functional models for efficient estimation of the expected feedback. The fitness critics not only provide expected feedback for policies, but push that policy feedback down to the agent's state-action pairs experienced throughout the policy's performance. The key insight into the effectiveness of fitness critics is to maximize information reuse. Fitness critics accomplish this by correlating the multiple instances where similar actions were taken from similar states regardless of in which policies they were used. In doing so, fitness critics generate a dense, state-action pair "value" function from the sparse, episodic policy evaluation.

The contributions of this paper are to:

- link the immediate policy feedback with the experienced state-action pairs to obtain a unique expected feedback for each state-action pair;
- obtain a policy-level fitness critic by aggregating the expected state-action feedback estimates;

These elements of fitness critics allow the EA to effectively train policies for tightly coupled domains.

We apply fitness critics in the tightly coupled multi-rover domain [23] to effectively train policies for the multiagent team. This paper shows that teams trained with fitness critics achieve comparable or increased mean performance scores compared to teams trained without fitness critics. Furthermore, only teams trained with fitness critics were able to demonstrate any amount of effective learning on tasks with higher degrees of coupling.

2 BACKGROUND

Learning in tightly coupled domains is difficult due to the inability to properly reinforce potentially contributing behaviors [23]. Potentially contributing behaviors that require coordination with other agents will not be contributing when there is no coordination. As a result, feedback from the environment, which is based on the performance of the team as a whole, does not accurately score the agent's potentially contributing behaviors. As such, tight coupling creates inconsistent feedback that makes it hard to reinforce potentially contributing rewards. The expected feedback (with respect to varied teams) is a more consistent value for evaluating an individual

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

agent's behavior as this feedback corrects for varying multiagent teams and performances.

Current methods that improve learning in multiagent systems are not as good at improving learning in tightly coupled domains. To better isolate an agent's contribution, effective multiagent reward shaping (Section 2.2) requires domain knowledge that may not be available. Sample-based methods (Section 2.3) require large amounts of samples in order to provide an acceptable expected feedback estimate. Regression methods (Section 2.3) map the expected fitness to policy parameters, which results in large functional models to train that does not leverage the information learned about one policy when evaluating other policies that exhibit similar behaviors.

The difficulties we address in this paper are:

- decentralized multiagent teams;
- tightly-coupling in feedback signal;
- sparse and delayed feedback;
- the available knowledge of domain being too limited to apply reward shaping.

2.1 Cooperative Coevolutionary Algorithms

Evolutionary algorithm (EAs) are optimization algorithms that update a population of genomes using the evolutionary operators: selection, recombination, mutation, and reinsertion [4]. For example, the weights for neural network policies can be treated as genomes to be evolved by the EA [8, 9]. Cooperative coevolutionary algorithms (CCEAs) extend evolutionary algorithms to multiagent domains by evolving agents' policies independently [29]. For each agent, CCEA evolves a separate population of policies. CCEAs can have been augmented with sampling-based methods (Section 2.3) like leniency [22] and hall of fame [24] to increase team performance [6].

2.2 Multiagent Reward Shaping

Reward shaping methods use known knowledge about the domain to augment the reward signal in order to make the reward signal more informative and improve the quality of learning. However, arbitrary reward shaping in the absence of domain knowledge may not preserve the optimal policy for a Markov decision process (MDP). Potential Based Reward Shaping (PBRS) [18] uses a potential function to shape rewards while preserving optima. However, there is still a need for domain knowledge to produce potential functions that have a positive effect on the quality of learning. For these methods, the *reward* is often step-wise feedback from the environment, but many of these approaches can be extended to also handle episodic feedback.

Difference evaluations (D) shapes the feedback to produce a feedback signal that isolates other agents' behaviors [3]. Difference evaluations replace the immediate feedback with new feedback that is the difference between what the original feedback is and an estimate of what the feedback would be without the agent's contribution. This is performed to isolate a more appropriate evaluation for the agent's contribution:

$$D_i = G(z) - G(z_{-i} \cup c_i) \tag{1}$$

where D_i is the agent i's difference evaluation; z is the collective state-action or sequence of state-actions for all agents; G is the team's evaluation; z_{-i} is the system state-action or sequence of state-actions for all agents excluding the evaluated agent i; c_i is a counterfactual state-action or sequence of state-actions that replaces those of agent i. Training with D has lead to higher performance in large-scale multiagent domains [7, 20, 28]. Although D may be used to measure the impact that an individual agent has on team performance, its effectiveness tends to decrease in applications where the tight coupling of agents' actions is necessary.

2.3 Methods for Estimating Expected Feedback

While methods exist to estimate the expected feedback for EAs, these methods become inefficient when dealing with the inconsistent feedback present in tightly coupled domains. Noisy fitness evaluations impede the EA's ability to optimize the expected fitness value. Leniency (i.e. taking the maximum of multiple fitness evaluation samples) [22], hall of fame (i.e. optimizing the best team), averaging multiple fitness evaluations [5], and increasing the evolutionary algorithm's population size [12], are among some of the sampling-based methods for minimizing the negative effect of noise in the fitness evaluation. These methods can be inefficient when the sampling of good actions is difficult.

Fitness approximation methods will regress a functional model, and use that functional model to provide informed estimates of the expected feedback, thereby avoiding the need to perform then average multiple performance evaluations for each new estimate. These methods, such as the Memory-based Fitness Evaluation Genetic Algorithm [25], or approximating the fitness function using regression [21], estimates the expectation of the original fitness function. In this way, information from previous fitness evaluations is conserved in the functional model. Nevertheless, by mapping expected feedback directly to policy parameters, these methods miss out on the opportunity to leverage information surrounding previously evaluated policies when evaluating differing policies that exhibit similar behaviors.

2.4 Actor-Critic Methods

The inspiration for fitness critics comes from the critic in the Off-policy Deterministic Actor-Critic (OPDAC) method [27]. The OPDAC method is an actor-critic method, which are methods that update a policy (or actor) by using the gradient of a functional model (or critic). The OPDAC method allows for variance-reduced training of deterministic policies in domains with continuous state-action spaces. Actor-critic methods and related reinforcement learning methods struggle with sparse and delayed rewards in continuous state-action spaces [10].

The training of a critic in actor-critic methods allows for variance reduction in the evaluations of state-action pairs. With this idea, fitness critics also train an interior functional model, or *intermediate critic*, but to provide sufficient expected feedback estimates. Thus, the intermediate critic is comparable to the critic from actor-critic methods in that they are both used to evaluate state-action pairs.

In contrast with the critics from actor-critic methods, fitness critics perform an aggregation step to create a singular evaluation for the policy, which is crucial, as the EA can only use a single fitness evaluation per episode. Another difference is that, with

fitness critics, the actor (in this case, the EA) performs gradient-free policy optimization by using the fitness critic's evaluation as opposed to performing gradient-based policy optimization using the critic's gradient.

3 FITNESS CRITICS

In a given episode, a multiagent team is executed in the environment and receives a score at the end of the episode based on their performance. The score serves as the *immediate feedback* for the evolutionary algorithm (EA). In multiagent domains, reward shaping methods (Section 2.2), like difference evaluations, can be applied to the immediate feedback to provide each agent with a unique, and individualized shaped feedback. However, even then, this feedback may not properly reflect the potential contribution of an individual agent. Instead, the EA may use the individual agent's *expected feedback* (with respect to varied teams) to better represent the agent's contribution and improve the quality of learning. However, the expected feedback is not always accessible and must instead be estimated. Efficient estimation of the expected feedback is not a trivial task.

Fitness critics are functional models for efficiently estimating the expected feedback. The key insight into the effectiveness of fitness critics is to maximize information reuse. Fitness critics do this by correlating the multiple instances where the agent took similar actions in similar states regardless of which policy the agent used. In the multiagent system, each agent has its own fitness critic.

To provide the expected feedback estimates, the fitness critic updates, and makes use of an interior functional model called the *intermediate critic*, which is trained simultaneously with the execution of the EA. The fitness critic then uses the intermediate critic to evaluate each state-action pairs in the agent's *trajectory* (i.e. the sequence of experienced state-action pairs for an episode). We provided further details on fitness critic evaluations in Section 3.1. The fitness critics aggregate these evaluations and use the result of the aggregation as the estimation of the expected feedback.

The fitness critic uses feedback data to perform the aforementioned updates on the intermediate critic (Sections 3.2 and 3.3 provide the details about these updates). Throughout the training episodes, our method stores this feedback data in an experience replay buffer [13] to build a mapping between the agent's *trajectories* and the immediate feedback associated with each trajectory. With each update of the intermediate critic, the fitness critic, as a whole, gradually learns this feedback data mapping.

Figure 1 shows a multi-step illustration of fitness critics in action.

3.1 Evaluating with Fitness Critics

For each step of an episode, every agent in the team receives an input state from the environment and responds with an action to interact with the environment. This process is repeated for many steps until the end of the episode. In each step i, the intermediate critic evaluates the state-action pair, and generates an evaluation c_i:

$$c_i = C(\mathbf{s_i}, \mathbf{a_i}; \mathbf{w}) \tag{2}$$

where the function C is the intermediate critic that is parameterized by the vector $\mathbf{w}$; and $\mathbf{s_i}$ and $\mathbf{a_i}$ are the state and action, respectively. The fitness critic then aggregates the many intermediate critic evaluations to provide an expected feedback estimate. In this paper, we use the maximum or the mean functions for aggregation:

$$F_{max}(T) = \max_{(\mathbf{s},\mathbf{a})\in T} C(\mathbf{s}, \mathbf{a}) \tag{3}$$

$$F_{mean}(T) = \frac{\sum_{(\mathbf{s},\mathbf{a})\in T} C(\mathbf{s}, \mathbf{a})}{\sum_{(\mathbf{s},\mathbf{a})\in T} 1} \tag{4}$$

$$T = \{(\mathbf{s}_1, \mathbf{a}_1), ..., (\mathbf{s}_n, \mathbf{a}_n)\} \tag{5}$$

where (F_{max}) is the fitness critic using the maximum function for aggregation; (F_{mean}) is the fitness critic using the mean function for aggregation; and T is the evaluated *trajectory*, which is the sequence of state-action pairs $(\mathbf{s}_i, \mathbf{a}_i)$ experienced throughout an episode with n steps.

3.2 Fitness Critics Update

Throughout an agent's learning process, the intermediate critic is periodically updated. These updates allow the fitness critic to learn the stored mapping between agent trajectories and the immediate feedback for these trajectories. The intermediate critic is updated via the fitness critic using mini-batch gradient descent. These updates are performed to minimize the expected mean squared error between the fitness critic evaluation and the immediate feedback for a trajectory:

$$\mathbf{w}' = \mathbf{w} + \Delta_{\mathbf{w}} \tag{6}$$

where $\mathbf{w}'$ is the post-update fitness critic parameter vector; and $\mathbf{w}$ is the pre-update fitness critic parameter vector; $\Delta_{\mathbf{w}}$ is $\mathbf{w}$'s update, given by:

$$\Delta_{\mathbf{w}} = \alpha_{\mathbf{w}} \sum_{k=1}^{n_s} \delta_k \nabla_{\mathbf{w}} F^{\mathbf{w}}(T_k) \tag{7}$$

$$\delta_k = f_k - F^{\mathbf{w}}(T_k) \tag{8}$$

where $\alpha_{\mathbf{w}}$ is the learning rate; n_s is the number of samples for every mini-batch; k is the sample index; the function $F^{\mathbf{w}}$ is the fitness critic that is parameterized by $\mathbf{w}$; and δ_k is the error between the experienced immediate feedback f_k for the sample trajectory T_k and fitness critic's evaluation for T_k.

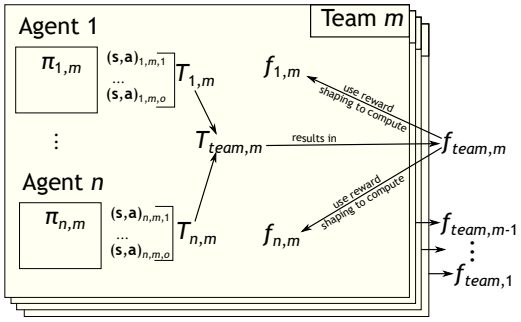
3.3 Inexact Fitness Critics Updates

For the max fitness critic (F_{max}), the only state-action pair in a trajectory that contributes to the parameter update is the state-action pair that receives the maximum intermediate critic evaluation for that trajectory. All other state-action pairs do not affect the update, as these state-action pairs provide zero derivatives due to the nature of the maximum function. This limits how fast the fitness critic learns. Instead, the fitness critic's can be updated quickly (but at the cost of accuracy) by associating every state-action pair in a trajectory with the immediate feedback. Subsequently, the intermediate critic directly learns a new feedback data mapping between state-action pair and immediate feedback:

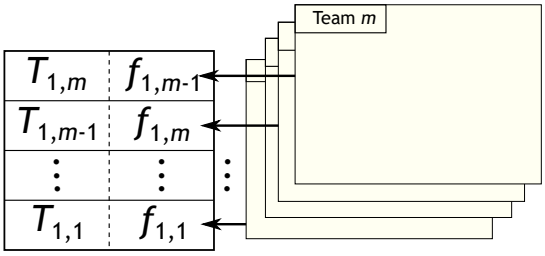
$$\Delta_{\mathbf{w}} = \alpha_{\mathbf{w}} \sum_{K=1}^{n_s} \delta_k \nabla_{\mathbf{w}} C^{\mathbf{w}}(\mathbf{s}_k, \mathbf{a}_k) \tag{9}$$

$$\delta_k = f_k - C^{\mathbf{w}}(\mathbf{s}_k, \mathbf{a}_k) \tag{10}$$

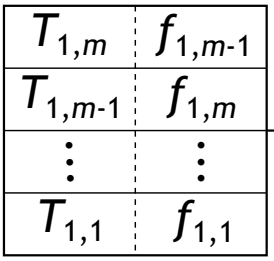
where C is the intermediate critic parameterized by $\mathbf{w}$; $\mathbf{s}_k$ and $\mathbf{a}_k$ are, respectively, the sampled state and action that were experienced



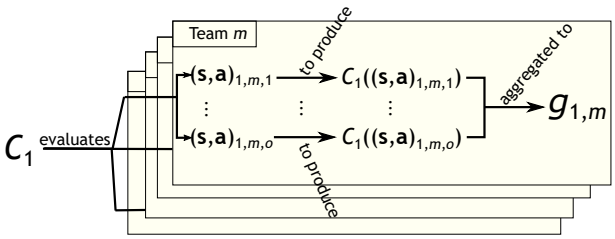
(a) A team m of n agents executes their policies $(\pi_{1,m}, \ldots, \pi_{n,m})$. The team's joint-trajectory $T_{team,m}$ receives a score of $f_{team,m}$. Reward shaping can be applied to $f_{team,m}$ to generate shaped feedback values $(f_{1,m}, \ldots, f_{n,m})$. This process is repeated for all the m episodes in each evolutionary algorithm (EA) generation.



(b) Our method maps Agent 1's trajectories $(T_{1,m}, \ldots, T_{1,1})$ for each of m episodes to the shaped feedback for that episode. Our method repeats this process for all other agents.



(c) Once per generation, our method uses the mapping between the trajectories and shaped feedback to update Agent 1's intermediate critic (C_1), which is a functional model. Our method repeats this process for all other agents.



(d) Agent 1's intermediate critic C_1 evaluates experienced state-action pairs in the $T_{1,m}$ trajectory $((\mathbf{s}, \mathbf{a})_{1,m,1}, \ldots, (\mathbf{s}, \mathbf{a})_{1,m,o})$. These evaluations are aggregated to give a fitness critic evaluation of $g_{1,m}$, which the EA uses as $\pi_{1,m}$'s fitness value. Our method repeats this process for all other episodes and all other agents.

Figure 1: Fitness critics in action.

together in a step of a past episode. Note that the fitness critic as a whole F and the intermediate critic C share the parameters $\mathbf{w}$ as F is just C with aggregation. With inexact fitness critic updates, (9) replaces (7) and (10) replaces (8).

3.4 Implementation of Fitness Critics in Cooperative Coevolutionary Algorithms

Our method "intercepts" the immediate feedback that the EA would otherwise have used as the policy's fitness value. Instead, our method stores the immediate feedback with the trajectory in the feedback data mapping. The fitness critic then evaluates the policy's trajectory with (3) for the max fitness critic, or (4) for the mean fitness critic, to provide an expected feedback estimate. The EA will use this value as the policy's fitness value when performing evolutionary operations. Figure 2 shows the interactions between the EA, the fitness critic, and the agents. At the end of each generation, our method updates the fitness critic with (6), (7) and (8) for regular updates; or (6), (9) and (10) for inexact updates. Algorithm 1 describes an application of fitness critics in the Cooperative Coevolutionary Evolutionary Algorithm (CCEA). We highlight the fitness critic additions to the base CCEA algorithm.

4 EXPERIMENTATION

We compare the performance of multiagent teams trained with and without fitness critics on the tightly coupled multi-rover domain. The policies for teammates are randomly selected so that there is diversity in the feedback signal. To evaluate the comparative advantage of using fitness critics, we also compare to the mean performance scores for teams trained with the average of the feedback for 10 diverse episodes. The EA uses this average as a simple, expected feedback estimate.

4.1 The Tightly Coupled Multi-Rover Domain

The tightly coupled multi-rover domain [23] is a domain where a team of rovers with limited sensing capabilities has the task of capturing various points of interest (POIs). In this domain, each POI has a value that determines the reward for capturing it. The

Algorithm 1 Cooperative Coeveolutionary Evolutionary Algorithm with Fitness Critics

Given n agents
Set k as the number of policies to generate per agent
for all *Agents* i **do**
  Initialize a population of policies $\rho_i \leftarrow \{\mu_{i,1}, ..., \mu_{i,k}\}$
  Initialize fitness critic $F_i^{\mathbf{w}}$
  Initialize experience replay buffer E_i
end for
for all *Generations* **do**
  for all *Training Episodes* **do**
    for all *Agents* i **do**
      Get (random) policy μ_i from ρ_i
    end for
    Assemble team $\mu_{team} \leftarrow \{\mu_1, ..., \mu_n\}$
    Execute μ_{team}
    Receive trajectories $T_{team} \leftarrow \{T_1, ..., T_n\}$
    for all *Agents* i **do**
      Receive immediate feedback f_i for T_i given T_{team}
      Set μ_i's fitness value g_i to $F_i^{\mathbf{w}}(T_i)$
      Update E_i given T_i and f_i
    end for
  end for
  for all *Agents* i **do**
    Update ρ_i with evolutionary operators
    Update $F_i^{\mathbf{w}}$ in batches with random samples from E_i
  end for
end for

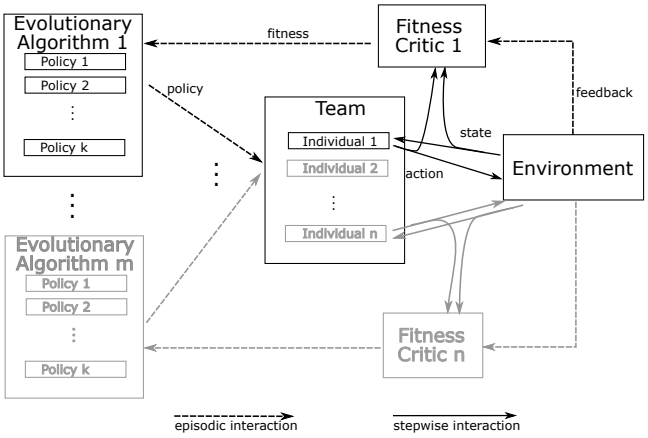


Figure 2: Interactions between the agent, the environment and the fitness critic in the cooperative coevolutionary algorithm for a multiagent system with n agents. Each agent evolves k policies. At the end of the episode, the fitness critic intercepts the feedback from the environment. The fitness critic then assigns a fitness evaluation to the agent's policy based on the agent's trajectory.

performance score for the team increases with the number of POIs captured. The number of rovers required for capturing a POI, n_{req}, represents the degree of coupling for the task. With higher degrees of coupling, the difficulty of the domain increases as untrained agents are less likely to stumble upon and reinforce coordinated behaviors. For simplicity, we do not simulate collision in this domain.

4.1.1 Scoring Team Performance. The team score G is a cumulative function of the POIs captured during the episode. At leastn_{req} rovers must be simultaneously within a POI's capture radius to capture that POI. The domain reports the team score at the end of the episode:

$$G = \sum_{k \in K} \max_t \mathbb{I}_{k,t} V_k \tag{11}$$

where V_k is the value of the POI k in the set of all POIs K; and $\mathbb{I}_{k,t}$ is an indicator function for whether there was any successful capture of k at some time step t:

$$\mathbb{I}_{k,t} = \begin{cases} 1 & \text{if } \sum_{i \in I} \mathbb{J}_{i,k,t} \geq n_{req} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\mathbb{J}_{i,k,t} = \begin{cases} 1 & \text{if } \delta_{i,k,t}^2 \leq d_{cap}^2 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where $\mathbb{J}_{i,k,t}$ is an indicator function that returns whether the rover i was within the capture radius of POI k at some time step t; I is the the set of all rovers; and $\delta_{i,k,t}$ is a bounded euclidean distance metric that is given by:

$$\delta_{a,b,t} = \max(||\mathbf{p_{a,t}} - \mathbf{p_{b,t}}||, d_{min}) \tag{14}$$

where a and b are two objects, with each object being either a rover or a poi; $\mathbf{p_{a,t}}$ and $\mathbf{p_{b,t}}$ are, respectively, the positions of a and b at time step t; and d_{min} is the lower bound for this distance metric.

The multiagent system can further shape the team score G with reward shaping to provide individual rovers with unique, and informative feedback. In this paper, we shape the team score with difference evaluations (see Section 2.2).

4.1.2 Rover Sensing and Motion. The policies that control the rovers are decentralized (i.e. each rover's policy is executed independently, based on each individual rover's local perception of the world). Each policy receives an 8-dimension vector from the rover's sensors as its state input and returns a 2-dimension vector describing the rover's motion. The rovers sense and move according to their orientation. Figure 3 shows the rovers' sensing and moving capabilities.

Each component of the state vector encodes relational information between the sensing rover and the POIs, or other rovers, in one of four quadrants centered around the sensing rover. The rover sensing function $s_{rover,q}$ is:

$$s_{rover,q} = \sum_{j \in J_q} \frac{1}{\delta_{i,j,t}^2} \tag{15}$$

where q is quadrant for the sensing value; and $\delta_{i,j,t}$ is the current bounded distance between the sensing rover i and some other rover j; J_q is the set of all rovers in q.

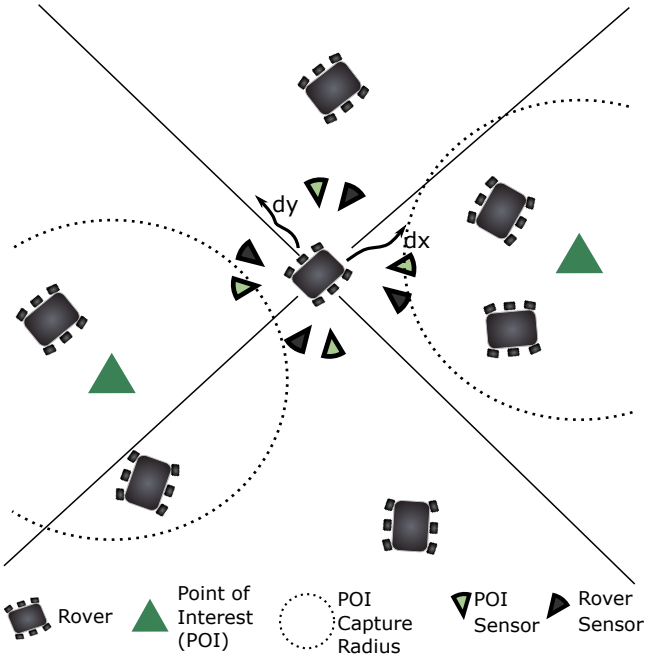


Figure 3: Rovers in the multi-rover domain move and sense according to their orientation. Rovers have a total of 8 sensors: a set of four quadrant sensors for sensing point of interests (POIs) and another four quadrant sensors for sensing other rovers.

The POI sensing function is $s_{poi,q}$:

$$s_{poi,q} = \sum_{k \in K_q} \frac{V_k}{\delta^2_{i,k,t}} \tag{16}$$

where $\delta_{i,k,t}$ is the current bounded distance between the sensing rover i and some POI k; K_q is the set of all POIs in q; and V_k is the value of the POI k.

4.2 Experimental Parameters

There are 4 POIs in the experimental domain. Each POI has a different unique value in the range of [1,4]; thus, the maximum team performance score for an episode is 10. We initialize the POIs on the circumference of a circle with a radius of 15. We initialize the rovers randomly in a concentric circle with a radius of 1.5. We set the rovers' initial orientation randomly. We perform this initialization scheme for the rovers and POIs to put the POIs far enough from the rovers, preventing the rovers from easily capturing the POI with untrained policies. However, rovers will still have enough time to reach to the POIs and reinforce coordinated behaviors.

The agent receives the difference evaluation as the immediate feedback for offline training. The difference evaluation is estimated by removing the recorded trajectory of the evaluated agent and calculating the resulting difference in G; d_{min} is set to 1; d_{cap} is set to 4.

The number of rovers required to capture a POI (n_{req}), which is varied from 3 to 6, represents the degree of coupling for a domain is. We also vary the number of rovers in total to investigate how well fitness critics respond to different team sizes, There are 50 steps in an episode.

Each agent has an evolving population of 50 neural network policies that are trained by the CCEA for over 5000 generations, which is enough to the team performance to convergence. Each neural network is a feed-forward and fully connected network with one hidden layer. Each network has 8 inputs for rover states, 32 hidden neurons, and 2 outputs for rover relative motion (Section 4.1.2 defines the rover states and motion). The hidden neurons use ramp activation over sigmoid activation for computational efficiency. However, the output neurons use the hyperbolic tangent as a sigmoid activation function to simulate motor saturation.

The CCEA uses binary tournament for selection to allow for diversity in the evolving population. The CCEA duplicates the selected neural networks to produce offsprings and then mutates the offsprings before putting them back into the population. Offspring mutation occurs with a mutation rate of 0.01 per weight by adding a sample from the standard Cauchy distribution to each mutated neural network weight.

The intermediate critic is a feed-forward and fully connected neural network with one hidden layer. Each critic has 10 inputs (the sum of state and action dimension), 80 hidden neurons and 1 output (for the state-action pair evaluations). For computational efficiency, the hidden neurons use leaky ramp activation [16] with a leaky scale value of 0.01. The output neurons use linear activation. To avoid vanishing gradients, the weights between the hidden layer and the output layer were fixed to 1, not including the bias weights. Fixing these weights do not negatively impact the representation capacity of the neural network.

We compare the effects of max critics with inexact updates and mean critics with regular updates on the multiagent team performance. We set the mini-batch size for the fitness critic updates and the number of mini-batches per generation to loosely match the number of feedback data entries generated in every generation. The replay buffer size is roughly 10 times the product of these two numbers to preserve some information from prior generations. For the max fitness critics with inexact updates, the replay buffer size is 25000 and mini-batch size is 50, with 50 mini-batches per generation. For the mean critics with regular updates, the replay buffer size is 500 and the mini-batch size is 7, with 7 mini-batches per generation. The reason for the differences in these values between the fitness critics is to compensate for the different update methods (regular/inexact) and to have each critic consume similar amounts of computing resources. The learning rate is 5×10^{-6}.

5 RESULTS

We compare the mean performance score of multiagent teams trained with and without fitness critics on the tightly coupled multi-rover domain with 15 rovers, 4 POIs, and various degrees of coupling (n_{req}). All teams were able to demonstrate effective learning on the task with $n_{req} = 3$. However, using the immediate feedback, teams achieved poor scores on the task with $n_{req} = 4$, and were ineffective at learning on the tasks with $n_{req} = [5, 6]$. Use the average feedback of 10 episodes as an estimate for the expected feedback, the teams' scores on the task with $n_{req} = [4, 5]$ improved, but the teams were still ineffective at learning on tasks with $n_{req} = 6$. We also ran

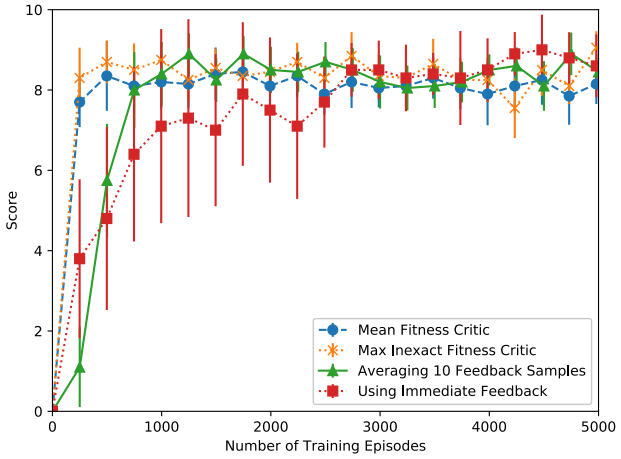


Figure 4: Mean performance score curves for teams of agents trained with and without fitness critics. Agents are trained on the tightly-coupled multi-rover domain with 15 rovers, 4 points of interests (POIs), and the degree of coupling (n_{req}) set to 3. All teams perform similarly. Error bars denote 95% confidence interval.

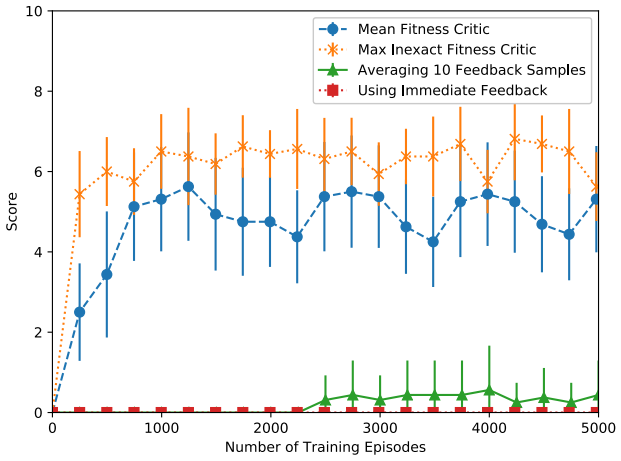


Figure 5: Mean performance score curves for teams of agents trained with and without fitness critics. Agents are trained with CCEA on the tightly-coupled multi-rover domain with 15 rovers, 4 points of interests (POIs), and the degree of coupling (n_{req}) set to 5. Teams trained with fitness critics outperform other teams. Error bars denote 95% confidence interval.

experiments with a comparable instance of *multi-agent deep deterministic policy gradient* (MADDPG) [15], an actor-critic algorithm. MADDPG was able to learn on the task with ($n_{req} = 1$) but failed to learn for any of the aforementioned tasks with higher degrees of coupling. MADDPG's failure to learn on these tasks might be due to its inability to handle the sparse reward signal in the domain as a result of tight coupling.

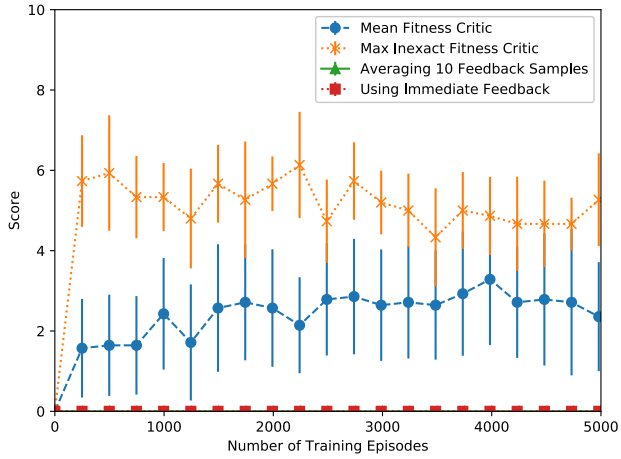


Figure 6: Mean performance score curves for teams of agents trained with and without fitness critics. Agents are trained with CCEA on the tightly-coupled multi-rover domain with 15 rovers, 4 points of interests (POIs), and the degree of coupling (n_{req}) set to 6. Teams trained with fitness critics outperform other teams, which are ineffective at learning. Error bars denote 95% confidence interval.

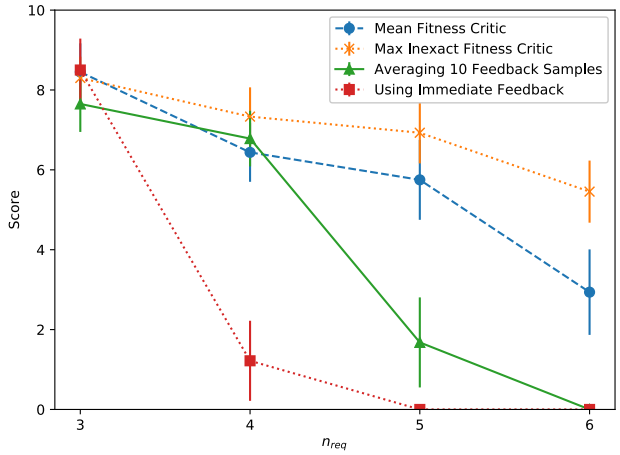


Figure 7: Mean performance scores for teams of agents trained with and without fitness critics. Agents are trained with CCEA on the tightly-coupled multi-rover domain with 15 rovers, 4 points of interests (POIs), and various degrees of coupling (n_{req}). Error bars denote 95% confidence interval.

Teams trained with fitness critics performed comparable or better than other teams. These teams demonstrate effective learning on all tasks, though teams trained with the max critic with inexact updates outperformed teams trained with the mean critic with regular update. We suspect that optimistic exploration provided by the maximum aggregation function in the max critic contributes to the success of the max critic as optimistic exploration allows

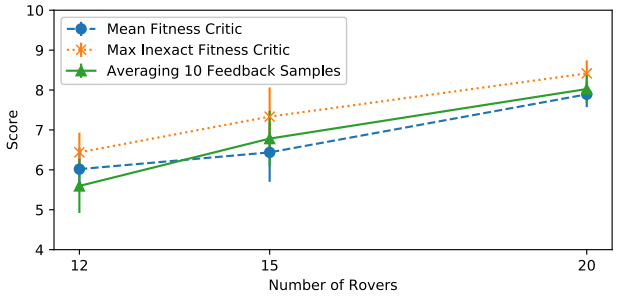


Figure 8: Mean performance scores for teams of agents trained with and without fitness critics. Agents are trained with CCEA on the tightly-coupled multi-rover domain with a varying number of rovers, 4 points of interests (POIs), and a degree of coupling (n_{req}) set to 4. All teams perform similarly. Error bars denote 95% confidence interval.

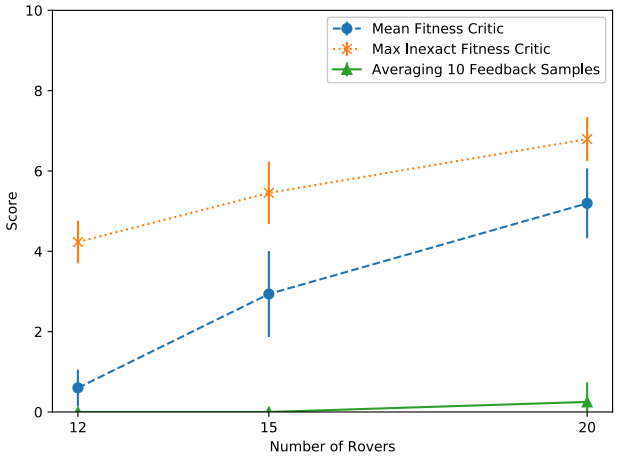


Figure 9: Mean performance scores for teams of agents trained with and without fitness critics. Agents are trained with CCEA on the tightly-coupled multi-rover domain with a varying number of rovers, 4 points of interests (POIs), and a degree of coupling (n_{req}) set to 6. Teams trained with fitness critics outperform teams trained with an average of feedback samples. Error bars denote 95% confidence interval.

for agents to stumble upon rare but highly-valued state-action spaces more often. We illustrate these results on the performance of teams trained with and without fitness critics on tasks with various degrees of coupling in Figures 4 to 7.

We also compare the mean performance scores of teams trained with fitness critics to teams trained with the average of 10 feedback samples on the tightly coupled multi-rover domain with various team sizes on the tasks with $n_{req} = 4$ and $n_{req} = 6$. Again, teams trained with fitness critics performed comparable or better. We also notice an increase in the achieved mean performance scores as the number of rovers increases which may be due to having more rovers to explore and find the POIs. Figures 8 and 9 present these results.

Teams trained with fitness critics are able to outperform other teams due to the fitness critics' ability to store feedback information of past generations and readily use that information for future evaluations. Reusing feedback information in this way allows fitness critics to be more efficient and accurate at providing expected feedback estimates.

6 CONCLUSION

In tightly coupled domains, sparse *immediate feedback* can be uninformative and lead to ineffective learning. The *expected feedback* estimate allows for more efficient training of decentralized multiagent teams with evolutionary algorithms (EAs) as the expected feedback (with respect to varied teams) prevents an agent from receiving bad feedback on a potentially good action in cases where they have poorly performing teammates. However, the expected feedback is not always accessible and must instead be estimated. Efficient estimation of the expected feedback is not a trivial task.

This paper introduces *fitness critics*, a functional model for efficient estimation of the expected feedback. Fitness critics can provide this estimate efficiently by leveraging the information that is learned about one policy when evaluating other policies that exhibit similar behaviors. To do so, fitness critics link the immediate feedback policy with the state-action pairs that the agent experiences during the execution of the policy. Fitness critics then create expected feedback estimates for each of these state-action pairs and aggregate these estimates to obtain a policy-level fitness critic.

We apply fitness critics in the tightly coupled multi-rover domain to effectively train policies for the multiagent team. This paper shows that teams trained with fitness critics achieve comparable or increased mean performance scores compared to teams trained without fitness critics. Furthermore, only teams trained with fitness critics were able to demonstrate any amount of effective learning on tasks with higher degrees of coupling. Fitness critics have a positive effect on learning in tightly coupled domains due to their ability to efficiently filter noise from the feedback signal by providing expected feedback estimates.

Fitness critics is a new addition to a broader collection of work on marrying evolutionary algorithms (EAs) with reinforcement learning (RL) to address each others' deficiencies. Related to this work, memetic algorithms [11] extend EAs with local search techniques for more efficient convergence to good solutions. Also, there is Evolutionary Reinforcement Learning [10], which combines the two learning methods to provide sample efficient learning. Future work with fitness critics will investigate the potential in using the fitness critic to perform a reinforcement-like policy gradient, as done in actor-critic methods, in addition to EA, to further improve learning efficiency.

7 ACKNOWLEDGEMENT

This work was partially supported by the Air Force Office of Scientific Research under grant No. FA9550-19-1-0195 and the National Science Foundation under grant No. IIS-1815886. The authors thank Stephane Airiau for his thoughtful feedback on this paper.

REFERENCES

[1] Adrian Agogino, Chris HolmesParker, and Kagan Tumer. 2012. Evolving Large Scale UAV Communication System. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO '12)*. Association for Computing Machinery, New York, NY, USA, 1023–1030. https://doi.org/10.1145/2330163.2330306

[2] Adrian Agogino and Kagan Tumer. 2007. Evolving Distributed Agents for Managing Air Traffic. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07)*. ACM, New York, NY, USA, 1888–1895. https://doi.org/10.1145/1276958.1277339

[3] A. Agogino and K. Tumer. 2008. Efficient Evaluation Functions for Evolving Coordination. *Evol. Comput.* 16, 2 (June 2008), 257–288. https://doi.org/10.1162/evco.2008.16.2.257

[4] Thomas Back, David B. Fogel, and Zbigniew Michalewicz (Eds.). 1999. *Basic Algorithms and Operators* (1st ed.). IOP Publishing Ltd., Bristol, UK, UK.

[5] Erick Cantú-Paz. 2004. Adaptive Sampling for Noisy Problems. In *Genetic and Evolutionary Computation – GECCO 2004*, Kalyanmoy Deb (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 947–958.

[6] Mitchell Colby and Kagan Tumer. 2012. Shaping Fitness Functions for Coevolving Cooperative Multiagent Systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS '12)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 425–432. http://dl.acm.org/citation.cfm?id=2343576.2343637

[7] Mitchell Colby, Logan Yliniemi, and Kagan Tumer. 2016. Autonomous Multiagent Space Exploration with High-Level Human Feedback. *Journal of Aerospace Information Systems* 13, 8 (2016), 301–315. https://doi.org/10.2514/1.I010379 arXiv:https://doi.org/10.2514/1.I010379

[8] Verena Heidrich-Meisner and Christian Igel. 2009. Neuroevolution Strategies for Episodic Reinforcement Learning. *J. Algorithms* 64, 4 (Oct. 2009), 152–168. https://doi.org/10.1016/j.jalgor.2009.04.002

[9] C. Igel. 2003. Neuroevolution for reinforcement learning using evolution strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, Vol. 4. 2588–2595 Vol.4. https://doi.org/10.1109/CEC.2003.1299414

[10] Shauharda Khadka and Kagan Tumer. 2018. Evolution-guided Policy Gradient in Reinforcement Learning. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., USA, 1196–1208. http://dl.acm.org/citation.cfm?id=3326943.3327053

[11] Natalio Krasnogor. 2008. An Unorthodox Introduction to Memetic Algorithms. *SIGEVOlution* 3, 4 (Dec. 2008), 6–15. https://doi.org/10.1145/1621943.1621945

[12] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., USA, 6405–6416. http://dl.acm.org/citation.cfm?id=3295222.3295387

[13] Long-Ji Lin. 1992. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Mach. Learn.* 8, 3-4 (May 1992), 293–321. https://doi.org/10.1007/BF00992699

[14] Daniele Loiacono. 2012. Learning, Evolution and Adaptation in Racing Games. In *Proceedings of the 9th Conference on Computing Frontiers (CF '12)*. ACM, New York, NY, USA, 277–284. https://doi.org/10.1145/2212908.2212953

[15] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent Actor-critic for Mixed Cooperative-competitive Environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., USA, 6382–6393. http://dl.acm.org/citation.cfm?id=3295222.3295385

[16] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. 3.

[17] David E Moriarty and Risto Miikkulainen. 1995. Discovering Complex Othello Strategies Through Evolutionary Neural Networks. *Connection Science* 7, 3 (1995), 195–209.

[18] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 278–287. http://dl.acm.org/citation.cfm?id=645528.657613

[19] Stefano Nolfi and Dario Floreano. 2000. *Evolutionary Robotics: The Biology,Intelligence,and Technology*. MIT Press, Cambridge, MA, USA.

[20] M. NoroozOliaee, B. Hamdaoui, and K. Tumer. 2013. Efficient Objective Functions for Coordinated Learning in Large-Scale Distributed OSA Systems. *IEEE Transactions on Mobile Computing* 12, 5 (May 2013), 931–944. https://doi.org/10.1109/TMC.2012.67

[21] I. Paenke, J. Branke, and Yaochu Jin. 2006. Efficient Search for Robust Solutions by Means of Evolutionary Algorithms and Fitness Approximation. *Trans. Evol. Comp* 10, 4 (Aug. 2006), 405–420. https://doi.org/10.1109/TEVC.2005.859465

[22] Liviu Panait, Karl Tuyls, and Sean Luke. 2008. Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective. *J. Mach. Learn. Res.* 9 (June 2008), 423–457. http://dl.acm.org/citation.cfm?id=1390681.1390694

[23] A. Rahmattalabi, J. J. Chung, M. Colby, and K. Tumer. 2016. D++: Structural credit assignment in tightly coupled multiagent domains. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4424–4429. https://doi.org/10.1109/IROS.2016.7759651

[24] Christopher D. Rosin and Richard K. Belew. 1997. New Methods for Competitive Coevolution. *Evol. Comput.* 5, 1 (March 1997), 1–29. https://doi.org/10.1162/evco.1997.5.1.1

[25] Y. Sano and H. Kita. 2002. Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 1. 360–365 vol.1. https://doi.org/10.1109/CEC.2002.1006261

[26] Jack F. Shepherd and Kagan Tumer. 2010. Robust Neuro-Control for a Micro Quadrotor. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*. Association for Computing Machinery, New York, NY, USA, 1131–1138. https://doi.org/10.1145/1830483.1830693

[27] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, I–387–I–395. http://dl.acm.org/citation.cfm?id=3044805.3044850

[28] K. Tumer and A. Agogino. 2009. Improving Air Traffic Management with a Learning Multiagent System. *IEEE Intelligent Systems* 24, 1 (Jan 2009), 18–21. https://doi.org/10.1109/MIS.2009.10

[29] R Paul Wiegand. 2003. *An Analysis of Cooperative Coevolutionary Algorithms*. Ph.D. Dissertation. George Mason University.