AN ABSTRACT OF THE DISSERTATION OF

Mitchell Kataichi Colby for the degree of Doctor of Philosophy in
Mechanical Engineering presented on April 2, 2014.

Title:

Theoretical and Implementation Improvements for Difference Evaluation Functions

Abstract approved: _____

Kagan Tumer

Multiagent learning with cooperative coevolutionary algorithms is a critical area of research, and is relevant to many real-world applications including air traffic control, distributed sensor network control, and game-theoretic applications such as border patrol. A key difficulty in multiagent learning is the credit assignment problem, where the impact of each individual agent on the overall system performance must be ascertained. Difference evaluation functions aim to solve this credit assignment problem, by approximating the effect that each agent has on the system evaluation function. Difference evaluations have proven to produce superior learned policies in many multiagent settings.

Although difference evaluations have produced excellent empirical results, there are still three key research questions that must be addressed regarding their usefulness in real-world systems. More specifically, the performance, theoretical

advantages, and methodology for implementation must be addressed in order to demonstrate that difference evaluations are practical for use in real-world multiagent learning. These research questions are addressed in this dissertation. The first contribution of this dissertation is to demonstrate that difference evaluations may be extended and combined with other coordination mechanisms, resulting in superior learned performance. The second contribution of this dissertation is to derive conditions which guarantee that difference evaluations will outperform traditional coordination mechanisms. The third and final contribution of this dissertation is to demonstrate that difference evaluations may be approximated using only local knowledge, allowing for their implementation in any generic multiagent learning setting. By addressing the performance, theoretical foundation, and implementation concerns of difference evaluations, this dissertation provides a detailed analysis demonstrating the usefulness of difference evaluation functions in multiagent learning systems.

Theoretical and Implementation Improvements for Difference Evaluation Functions

by

Mitchell Kataichi Colby

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 2, 2014
Commencement June 2014

Doctor of Philosophy dissertation of Mitchell Kataichi Colby presented on April 2, 2014.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Mitchell Kataichi Colby, Author

ACKNOWLEDGEMENTS

I would like to express my appreciation for my advisor, Dr. Kagan Tumer. This work would not have been possible without his support and guidance. I would also like to thank my parents, who have also pushed me towards engineering and science. I would like to thank all of my fellow students in the Autonomous Agents and Distributed Intelligence laboratory, as the conversations we had about research have always been immensely helpful.

I would like to express a special appreciation for my fiancee, Whitney. Working on my research has involved a lot of late nights and busy weekends, but she never complained and was always supportive. I would not have been able to complete this work without her support and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDICES

Chapter 1 – Introduction

1.1 Motivation

The automation of tasks through agent-based control is becoming more critical in real-world applications, as autonomous systems have significant advantages. For example, robots for planetary exploration, such as the Curiosity Rover, must also be autonomous. The rover is continuously exploring Mars terrain; if humans were required to constantly direct the rover, thousands of hours of manpower would be lost on a task that could be automated, and preventing those humans from working on other tasks. Further, by automating the task, more scientific research can be done due to elimination of lag from human directions. Ultimately, automation allows for agents to perform tasks without requiring human interaction; this allows humans to spend time on other tasks, increasing overall productivity.

Often, many autonomous agents are needed to achieve a complex task. For example, suppose there were 100 robots exploring Mars. Although one rover may explore Mars, many such rovers would be far more effective, allowing for a far greater amount of information to be collected and analyzed. In these *multiagent systems*, developing autonomous controllers becomes more difficult. Agents in a multiagent system interact with each other and the environment, and correct control decisions are often difficult to determine. For example, in the 100 Mars

rover setting, it is often unclear which area each rover should travel to in order to maximize the overall amount of data collected. Proper techniques to control and coordinate multiple agents must be developed in order to ensure efficiency and performance.

The control and coordination of large systems introduces a complexity well beyond the autonomy of a single agent. Though autonomy provides significant advantages for many applications such as self-driving cars, unmanned aerial systems, or household robots assisting with the elderly, it is often impossible for a human to determine how to control a large set of autonomous devices. There are simply too many agents to consider each individually, and coupling between agents is impossible to understand and predict for a human. Multiagent learning algorithms are designed to allow agents to learn policies, such that they may act autonomously but coordinate with other agents in a system in order for the collective of agents to achieve a system level goal.

Cooperative Coevolutionary Algorithms (CCEAs) involve training a set of autonomous agents to coordinate in order to achieve a system-level objective. Developing control policies for each agent in a multiagent systems is a critical area of research, and is involved in many real-world applications, including air-traffic control, distributed sensor network control, and multiple mobile robot control [13, 42, 48, 58, 84, 86, 116]. A key challenge to developing these control policies lies in addressing the *credit assignment problem*.

Multiple autonomous agents each follow a local control policy and interact with the environment and other agents in the system, making it difficult to determine

the effectiveness of a particular agent's control policy [4, 76, 94, 96]. The credit assignment problem must be addressed in order to develop adequate control policies for a multiagent system.

As a simple example of the credit assignment problem, consider the case of 100 mobile robots performing a surveillance task over some environment, and the goal is to collect as much information about the environment as possible. The problem which must be addressed is how to provide agent-specific feedback in order to determine the effectiveness of each agent's control policy. One option is to use the overall system level performance to provide feedback to each agent. However, this results in an unfavorable signal-to-noise ratio. A single agent cannot determine how it impacted the system performance, because 99 other agents are simultaneously impacting the system. Another option is to provide feedback to each individual agent based on that agent's amount of information collected. However, this feedback encourages individual agents to collect as much data as possible, which often results in agents competing over "easy to obtain" data and ignoring the "hard to obtain" data, ultimately harming overall system performance. In this case, agent objectives are not aligned with the system objective function, meaning agents acting to optimize their feedback actually decrease system performance. In order to address the credit assignment problem, agent feedback should have both a favorable signal-to-noise ratio, as well as be aligned with the system performance in order to ensure agents act to help the overall system.

One promising solution to the credit assignment problem is the *difference evaluation function*, which approximates an individual agent's impact on the system's

overall performance [3,7,118]. The difference evaluation function has two key theoretical properties which allow it to provide excellent agent-specific feedback. First, any agent which improves the value of its own difference evaluation also improves the overall system performance [5]. This alignment between an agent's local objective function and the system objective function ensures that learning agents are not encouraged to take actions which are detrimental to the overall system performance. Second, the difference evaluation function removes all portions of the system evaluation function which do not depend on the agent being evaluated. This results in a favorable signal-to-noise ratio in the agent's feedback signal, allowing the agent to more easily determine the impact of taking a particular action [5]. In addition to these theoretical properties, difference evaluation functions have produced excellent empirical results in a variety of multiagent systems, including congestion games, multiple robot coordination, and air-traffic control [2, 39, 120].

Although difference evaluation functions have useful theoretical properties in addition to extensive empirical results supporting their effectiveness, there are still many key research topics regarding difference evaluations that must be addressed. First, we must determine the *compatability* of difference evaluation functions with other coordination techniques. This allows us to determine what *performance gains* are available, in order to develop the best control policies possible. Second, we must conduct a *prescriptive theoretical analysis*, in order to determine under what conditions difference evaluations are expected to improve system performance. Third, we must develop a method to approximate difference evaluations, in order to *implement* them in systems where difference evaluations can not be directly computed.

1.2 Contributions

This dissertation addresses each of the three research topics described above. In particular, the contributions of this dissertation are to:

1. Present novel fitness assignment operators which utilize the difference evaluation function, and demonstrate the compatibility of difference evaluation functions with other coordination mechanisms, as well as the performance gains attainable with these pairings.

2. Provide an evolutionary game theoretic analysis of difference evaluations, and derive conditions under which they are expected to improve multiagent system performance.

3. Derive a novel method to approximate difference evaluation functions that relies on local state and action information as well as a broadcast value of the system performance.

The first contribution addresses the compatibility of difference evaluation functions with other coordination mechanisms, and how system performance is impacted through such combinations. Combining coordination mechanisms is often a difficult task, because different coordination mechanisms focus on different aspects of coordination. For example, difference evaluations focus on whether a particular agent could have performed better given a team of collaborating agents, while other coordination mechanisms such as leniency or hall of fame methods focus on whether a different team is better suited to collaborate with a particular agent. We

demonstrate that agent-centric and team-centric approaches to agent feedback are compatible, and in fact result in significant performance gains over either method used alone.

The second contribution addresses the theoretical conditions which guarantee that difference evaluations will improve performance when compared to traditional coordination approaches, using an evolutionary game theoretic framework to derive these conditions. Multiagent learning systems are difficult to analyze theoretically, due to the high coupling between agents and the limited state information available to each agent. Very few prescriptive analyses have been conducted on multiagent learning techniques, and these techniques are often judged solely on their performance in a variety of empirical tests. This dissertation provides a prescriptive theoretical analysis of difference evaluation functions, deriving conditions which determine when difference evaluations provide benefits to the system and should thus be implemented.

The third contribution involves deriving a method to approximate difference evaluation functions when system and state information are limited. While difference evaluations provide good performance in multiagent learning systems, they are often difficult or impossible to directly compute in some systems. In order to implement difference evaluations to achieve improved performance, they often need to be approximated. This approximation is typically difficult, due to the global knowledge requirements of calculating difference evaluations. We demonstrate that difference evaluations may in fact be approximated using only local knowledge, allowing for their implementation in any generic multiagent system where the value

of the system evaluation function is broadcast.

This dissertation provides evidence demonstrating the effectiveness of difference evaluation functions; in particular, we demonstrate what performance gains can be achieved through the use of difference evaluation functions combined with other coordination mechanisms, derive conditions under which we expect to see these performance gains, and provide an approach to approximate difference evaluations in systems where global state knowledge is unavailable. By demonstrating compatibility and performance gains, providing a theoretical analysis, and providing a methodology for implementation, we demonstrate the real-world usefulness of difference evaluation functions.

The rest of this dissertation is organized as follows: Chapter 2 presents necessary background information and related work. Chapter 3 presents fitness assignment operators incorporating difference evaluation functions. Chapter 4 provides an evolutionary game-theoretic analysis of difference evaluation functions, and derives conditions under which they are expected to improve system performance. Chapter 5 presents a methodology for approximating difference evaluation functions. Finally, Chapter 6 concludes the dissertation.

Chapter 2 – Background

The following sections provide background information on evolutionary algorithms, cooperative coevolutionary algorithms, difference evaluation functions, fitness function shaping, evolutionary game theory, and evaluation function approximation.

2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of stochastic search algorithms, which have been shown to work well in complex domains where gradient information is not readily available [11, 20, 52, 74, 105, 133]. EAs typically contain three basic mechanisms: solution generation, mutation, and selection. These mechanisms act on an initial set of candidate solutions (the population) in order to generate new solutions and to retain solutions which show improvement [22, 80, 131]. A general evolutionary algorithm is shown in Algorithm 1. A population of k candidate solutions is initialized randomly. For each generation in the evolutionary algorithm, the following steps are taken. First, mutated copies (slightly altered) of each solution in the population are created, doubling the population size to $2k$. Then, each solution in the population is evaluated, and assigned a *fitness* related to that solution's performance. Finally, k solutions are selected for survival to

the next generation, with the probability of selection depending on the fitness of that solution. As long as the fitness operator captures the overall quality of the solution, and the selection operator favors higher fitness individuals, the quality of the solutions in the population gradually improves with respect to evolutionary time.

Algorithm 1: Evolutionary Algorithm

```
1  Initialize Population P with k solutions ;
2  foreach Generation do
3  |  foreach Population member p_i do
4  |  |  create mutated copy of p_i denoted p'_i ;
5  |  |  add p'_i to population P
6  |  end
7  |  foreach Population member p_i do
8  |  |  assign fitness f_i to agent p_i ;
9  |  end
10 |  select k members to survive based on fitness ;
11 end
```

EAs have shown excellent results for finding good solutions in single-agent settings [12,115,132], but need to be modified in order to apply to large multiagent search problems. One such modification is *coevolution*, which is detailed in the following section.

2.2 Cooperative Coevolutionary Algorithms

Cooperative Coevolutionary Algorithms (CCEAs) are an extension of EAs which are well-suited for multiagent domains [18, 31, 44, 49, 55, 88, 113, 123, 125, 129]. In a

CCEA, multiple populations evolve simultaneously in order to search for optimal policies for interacting agents [92, 107, 109, 122, 124, 126]. A general cooperative coevolutionary algorithm is given in Algorithm 2. A CCEA contains multiple coevolving populations, each of which corresponds to an agent in the system. At each generation, the following steps are executed. First, each population produces mutated solutions, just as in the standard evolutionary algorithm. Then, teams are created by drawing agents from each population (each agent is assigned to one team). The effectiveness of these teams are evaluated, and fitness is assigned to each agent on the team and then the agent is returned to its population. For each population, agents are selected to survive in the same manner as in standard evolutionary algorithms.

Algorithm 2: Standard CCEA

```
1  Initialize N populations of k solutions ;
2  foreach Generation do
3  |  foreach Population do
4  |  |  produce k successor solutions ;
5  |  |  mutate successor solutions
6  |  end
7  |  for i = 1 → 2k do
8  |  |  randomly select one agent from each population ;
9  |  |  add agents to team T_i ;
10 |  |  simulate T_i in domain ;
11 |  |  assign fitness to each agent in T_i
12 |  end
13 |  foreach Population do
14 |  |  select k solutions using ε-greedy
15 |  end
16 end
```

In a CCEA, the fitness of an agent depends on the interactions it has with other agents [23, 28, 75, 92, 113, 130]. The fitness of an agent acting in a team is somehow related to the overall team performance, which the agent cannot completely control. Thus, assessing the fitness of each agent is context-dependent and subjective [90]. Consider the example of two soccer-playing agents [91]. The first agent's task is to pass the ball to the second agent. The second agent must receive the pass and then shoot the ball into the goal. Success in this system is defined as a goal being scored. However, this system evaluation function does not provide clear agent-specific feedback. The first agent may make a perfect pass, but the second agent could either improperly receive the pass or miss the shot on goal. In this case, the first agent successfully completed its portion of the task, but the overall system goal was not met. Conversely, the second agent could be skilled at receiving passes and shooting, but if the first agent delivers a bad pass, then the second agent won't have the opportunity to score a goal. In either case, an agent with a "good" policy cannot determine that it took the correct actions based on feedback from the system evaluation function. Concepts such as leniency, the hall of fame, or difference evaluation functions all address this credit assignment problem, and are detailed in the following sections.

Clearly, credit assignment should not depend on the overall system performance, but on how well each individual agent completed its task. Solving this credit assignment problem is one of the primary concerns in cooperative coevolutionary algorithms, and this problem is addressed with fitness function shaping. We focus on shaping agent fitness values with the difference evaluation function.

2.3 Difference Evaluation Functions

The agent-specific Difference Evaluation Function $D_i(\vec{s}, \vec{a})$ has been empirically shown to be an effective fitness assignment operator in multiagent systems, and is given by [5, 6]:

$$D_i(\vec{s}, \vec{a}) = G(\vec{s}, \vec{a}) - G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i}) \tag{2.1}$$

where $G(\vec{s}, \vec{a})$ is the global evaluation function for a particular system state $\vec{s}$ and joint action $\vec{a}$, and $G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i})$ is the global evaluation function without the effects of agent i. The term $\vec{s}_{-i}$ is the system state without agent i; $\vec{a}_{-i}$ is the set of joint actions without agent i. The terms $\vec{c}_{s,i}$ and $\vec{c}_{a,i}$ are the *counterfactual* state and action, respectively, which are used to replace agent i, and must not depend on the state and action of agent i. This notation is used for brevity for the rest of this dissertation, but should be explained in detail. For an n-agent system, the state and action vectors are:

$$\vec{s} = \{s_1, s_2, ..., s_n\}$$
$$\vec{a} = \{a_1, a_2, ...a_n\}$$

The state and action vectors without the effects of agent i are:

$$\vec{s}_{-i} = \{s_1, s_2, ..., s_{i-1}, 0, s_{i+1}, ..., s_n\}$$
$$\vec{a}_{-i} = \{a_1, a_2, ..., a_{i-1}, 0, a_{i+1}, ...a_n\}$$

The counterfactual vectors contain the counterfactual action and value for agent i, and are zero padded elsewhere to allow for vector addition. They are defined as:

$$\vec{c}_{s,i} = \{0, 0, ..., 0, c_s^i, 0, ...0\}$$
$$\vec{c}_{a,i} = \{0, 0, ..., 0, a_s^i, 0, ..., 0\}$$

Intuitively, the difference evaluation function gives the impact of agent i on the global evaluation function, because the second term removes the portions of the global evaluation function not dependent on agent i. Note that in systems with a differentiable system evaluation function, it follows from Equation 2.1 that:

$$\frac{\partial G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i})}{\partial a_i} = 0 \tag{2.2}$$
$$\Rightarrow \frac{\partial D_i(\vec{s}, \vec{a})}{\partial a_i} = \frac{\partial G(\vec{s}, \vec{a})}{\partial a_i} \tag{2.3}$$

where a_i is the action taken by agent i. Thus, an agent acting to increase the value of $D_i(\vec{s}, \vec{a})$ will also act to increase the value of $G(\vec{s}, \vec{a})$. This property is termed *factoredness* [5]. Further, as the second term of $D_i(\vec{s}, \vec{a})$ removes the effects of all agents other than agent i, the difference evaluation function provides a feedback signal with much less noise than $G(\vec{s}, \vec{a})$. This property is termed *learnability* [5]. By being factored and learnable, the Difference Evaluation Function provides very effective agent-specific feedback for learning agents within a multiagent system. In the context of a CCEA, the difference evaluation function is a much better fitness assignment operator than the global evaluation function [39]. In the context

of multiagent reinforcement learning, the difference evaluation function provides accurate rewards for actions taken by individual agents [40]. The difference evaluation function has provided excellent results in many domains, including distributed sensor network control [40], rover control [6, 39], and air-traffic control [5].

Although there is theoretical and empirical evidence suggesting difference evaluations are beneficial in multiagent systems, there are three key research questions which must be addressed. First, it is unknown if difference evaluations are compatible with other coordination mechanisms, and whether combining difference evaluations with other coordination mechanisms improves system performance in CCEAs (as opposed to difference evaluations alone); such an analysis is necessary in order to provide the best system performance possible, and to provide evidence that difference evaluations can provide high enough performance for implementation in real-world systems. Second, there is no prescriptive theoretical analysis giving conditions under which difference evaluations are expected to improve system performance. This means there is no set of conditions that may be used to determine whether difference evaluations should be used. Third, computing difference evaluations is often difficult in practice. In order to compute $G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i})$, an agent must have knowledge of the global system state, the joint action taken by all agents, and the mathematical form of $G(\vec{s}, \vec{a})$. In practice, agents never have access to $G(\vec{s}, \vec{a})$ (or traditional optimization techniques would be used), and very rarely have access to global state and action information. This means that directly calculating the difference evaluation function is often impossible, and a technique for approximating these functions is necessary for real-world implementation.

In order to demonstrate that difference evaluation functions are useful tools for real-world applications, we must address the three research questions detailed above. Performance, theoretical benefits, and a path to implementation must all be demonstrated to provide insight on the real-world applicability of difference evaluation functions.

2.4 Fitness Function Shaping

The following sections describe multiple methods for shaping fitness functions in CCEAs. In addition to difference evaluation functions, we consider biased searches, leniency, shaping for alignment, evolving teams, and hall of fame methods.

Biasing Coevolutionary Search

Panait *et al.* [90] biased the evolutionary search in order to find optimal solutions, rather than becoming stuck at locally optimal points. In traditional coevolution, a single agent is rated on how well the team does as a whole; every agent in the team gets equal credit. This results in an unfavorable signal-to-noise ratio, as each agent is unable to determine its individual contribution to the team's performance. In a *Biased Cooperative Coevolutionary Algorithm* (BCCEA), the fitness of an individual is based partly on its interactions with other agents (as in usual CCEAs), and partly on an estimate of the best possible fitness for that individual if it is partnered with optimal collaborators. It is of note that optimal

collaborators are not optimal agents from a system-level perspective; rather, they are optimally suited to collaborate with an agent given that agent's particular policy. Biasing the agent's fitness in this manner gives an indication of the best possible system-level performance that particular agent can attain, given that the agents it collaborates with are optimized to collaborate with that agent. By biasing the coevolutionary search in this manner, the algorithm is better able to search for optimal policies, rather than stable policies, because each agent receives feedback related to its individual performance, rather than solely the team's performance. One issue with this approach is that estimating how an agent would perform with optimal collaborators is a nontrivial task, and becomes increasingly difficult in complex domains. To address the difficulty of analytically determining optimal collaborators, the concept of *leniency* was developed to approximate the set of optimal collaborators for an agent.

Leniency

Panait *et al.* [91] introduced the concept of *lenient* learners in coevolutionary algorithms, which are agents which forgive possible mismatched teammate actions that result in poor team performance. Using lenient learners in coevolution is shown to provide learners with more accurate information about their policies, which increases the likelihood of converging to an optimal solution. In practice, leniency is achieved by pairing agents with multiple sets of collaborators, and taking the highest team fitness achieved from all runs. This lowers the likelihood that a learn-

ing agent will receive poor feedback simply because it was paired with suboptimal teammates. Leniency provides a mechanism for approximating the set of optimal collaborators for an agent; by pairing an agent with multiple sets of collaborators and taking the highest fitness attained, leniency assigns fitness to agents based on their best-known possible performance.

Fitness Function Shaping

Hoen and De Jong shaped the utilities of the agents so as to contribute to the system evaluation, such that an agent maximizing its individual utility would act to increase the system evaluation. By shaping agent fitnesses, the learning process is sped up considerably [64]. This work is similar to that of Agogino and Tumer, and Knudson and Tumer, who utilized difference evaluations as fitness functions to evolve coordination in multiagent systems [5, 73]. By shaping fitness functions such that each agent's fitness is related to the individual's contribution to team performance, the signal-to-noise ratio is improved considerably, allowing for faster learning and better learned performance [41].

Evolving Teams

Coevolution is frequently a good algorithm to use in multiagent systems with heterogeneous agents. Haynes *et al.* used evolutionary algorithms to evolve *teams* of predators [63]. Rather than evolving single predators, one member of the pop-

ulation consisted of four predators. In this manner, the predators can evolve to cooperate. By evolving teams rather than individual agents, communication is not required in the domain; in place of communication, the team of predators evolves to act as if they know the other agents' future actions based on the state of the system [111] Though effective, this approach becomes computationally inefficient when the size of the team becomes large, as each population member in the evolutionary algorithm consists of a full set of cooperating agents. Further, although the result of the algorithm is a set of policies for a multiagent system, the process to create these policies is centralized. As a result, evolving entire teams typically results in slow convergence.

Hall of Fame

Rosin and Belew [101] introduced the concept of the *Hall of Fame* for competitive coevolution, in which top individuals are saved in order to test against in later generations. There are two reasons why it is beneficial to save these top individuals. First, keeping top individuals contributes genetic information to later generations, which is imperative when conducting any evolutionary algorithm. Secondly, by keeping top individuals, new individuals in later generations may be tested against the hall of fame members. We extended this concept to CCEAs by keeping hall of fame *teams*, rather than hall of fame *individuals* [39]. The fitness of an agent is then a weighted average of that agent's fitness when it performs with its team (as in standard coevolution), as well as its fitness when it is placed on the best performing

team known. In this manner, desirable genotypes won't be lost if they perform poorly for a few generations due to being paired with suboptimal collaborators in the coevolutionary algorithm.

Potential Based Reward Shaping

If improperly utilized, reward or fitness function shaping can lead to unintended and undesirable behaviors [47]. For example, when trying to make an agent learn to ride a bicycle from one point to another, an additional reward may be added for each timestep the bicycle stays balanced [47, 97]. However, the agent learned to exploit this reward by riding continuously without reaching its destination. Potential based reward shaping ensures that such problems do not occur, by calculating the difference between a potential function Φ defined over a source and destination state [47]:

$$PBRS = r + \gamma\Phi(s') - \Phi(s) \tag{2.4}$$

where r is the original reward, s is the source state, s' is the destination state, and γ is a discount factor. Potential based reward shaping is proven to not alter the optimal policy of an agent, while ensuring shaped rewards are not inappropriately exploited by an agent.

2.5 Evolutionary Game Theory

Evolutionary Game Theory (EGT) describes a set of models which can be used to analyze evolutionary processes, and are well suited for analyzing cooperative coevolutionary algorithms [90, 123]. EGT models populations of individuals who interact with each other in repeated trials, and these models are used to analyze how populations change with respect to evolutionary time [78, 103, 108]. Typically, EGT models assume that populations are infinite in size, and that expected fitness values are based on distributions over a finite number of strategies [17, 100]. The following sections define notation used in EGT settings, as well as provide the EGT models used to analyze coevolutionary algorithms.

2.5.1 Notation

We now describe the notation for a two population EGT model. This notation directly extends to systems with more than two populations, so we describe a two population system for brevity and clarity. Each population represents one agent in the system. If the first agent has a finite number of n distinct actions it can take, then its population at each generation is an element of $\Delta^n = \{p \in [0,1]^n | \sum_{i=1}^{n} p_i = 1\}$. A higher value x_i corresponds to a higher probability that the agent selects action i. If the second agent has m actions to choose from, then its population at each generation is an element of $\Delta^m = \{q \in [0,1]^m | \sum_{i=1}^{m} q_i = 1\}$. Assuming a symmetric system where both agents are equally rewarded, then the payoff matrix C is used to compute the fitnesses in one population, and C^T is used to calculate

fitnesses for the other population.

2.5.2 EGT Models

The two population evolutionary game theoretic model is defined as [90, 123]:

$$u_i^{(t)} = \sum_{j=1}^{m} c_{ij} y_j^{(t)} \tag{2.5}$$

$$w_j^{(t)} = \sum_{i=1}^{n} c_{ij} x_i^{(t)} \tag{2.6}$$

$$x_i^{(t+1)} = \left(\frac{u_i^{(t)}}{\sum_{k=1}^{n} x_k^{(t)} u_k^{(t)}} \right) x_i^{(t)} \tag{2.7}$$

$$y_j^{(t+1)} = \left(\frac{w_j^{(t)}}{\sum_{k=1}^{m} y_k^{(t)} w_k^{(t)}} \right) y_j^{(t)} \tag{2.8}$$

Equations 2.5 and 2.6 assign fitness to each population, while Equations 2.7 and 2.8 define the proportions of each population at the next time step. Thus, the EGT model defines how populations are assigned fitness and then change with respect to evolutionary time. This model can be used to analyze how populations in cooperative coevolutionary algorithms change, allowing for the performance of the algorithm to be analyzed. The EGT model may be used to analyze how difference evaluation functions affect population dynamics in cooperative coevolutionary algorithms.

2.6 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 [8, 15]. 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-specific evaluation functions to be easily shaped based on the overall system evaluation function [27,29,43,69,70,87,99,128]. The following sections describe different approaches for approximating system evaluation functions, as well as previous work involving approximating difference evaluations.

Function approximation is commonly seen in reinforcement learning applications, where the value function or system state is approximated due to limited state information [57, 67, 83]. 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, 16, 93]. 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 only partial state information is available to each agent.

Fitness functions have also been approximated for use in evolutionary algorithms [21, 79]. 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 [69, 85].

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 [95]. 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, etc.), 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. In cases where the components which comprise the system evaluation function are unknown, it is difficult to create an approximation based on domain knowledge. Second, this approach required global knowledge of the system; an approximation of the system evaluation function was developed using global state and action information, 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.

As seen above, there is a wide range of research involving approximation in

reinforcement learning and evolutionary algorithms. 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.

Chapter 3 – Shaping Fitness Functions with Difference Evaluation Functions

3.1 Motivation

Coordinating multiple agents[1] in order to achieve some system objective is an important area of research, and is critical in many domains including rover coordination, air traffic control, search and rescue, and unmanned aerial vehicle coordination [5, 119]. One approach to achieving coordination is the use of Cooperative Coevolutionary Algorithms (CCEAs), which involve evolving multiple populations simultaneously and evaluating the fitness of individuals based on the individual's interactions with other agents in the system [39, 89]. By evolving multiple populations at once, CCEAs project the search space into multiple, smaller, search spaces [26, 56, 123]. Each coevolving population in the CCEA searches through one of these projected spaces, resulting in a large amount of information available to agents being lost. Coevolving agents have only a fraction of the total state space available to them, and their fitness assignment is dependent upon how they perform when combined with agents from the other coevolving populations. Thus, CCEAs have the tendency to create agents which are capable of performing ade-

[1]This chapter based on "Shaping Fitness Functions for Coevolving Cooperative Multiagent Systems," M. Colby and K. Tumer, Proceedings of the 11th International Conference of Autonomous Agents and Multiagent Systems, 2012

quately with a wide range of collaborators, rather than specializing to perform well with the best set of collaborators; in other words, CCEAs often produce stable, rather than optimal solutions [19, 49, 90]. In order to ensure CCEAs are a viable option for coordinating multiagent systems, it is critical that steps be taken to achieve optimal coordination policies.

There have been multiple approaches to address the issues of suboptimal stable policies. One approach involves shaping local fitness functions to align with the system evaluation function. Proper shaping of these fitness functions leads to faster learning and more optimal policies [5, 60, 64, 81, 112]. Another approach is to bias searches based on the notion of optimal teammates [90], which involves estimating agent utilities as if they were paired with optimal collaborators. Although these biased searches generally increase the effectiveness of CCEAs, an issue with this approach is that in complex domains, estimating system evaluations as if optimal collaborators were present is exceedingly difficult. Other methods involve altering the evolutionary mechanisms in CCEAs in order to optimize CCEA performance, such as lenient learners or hall of fame methods [91, 101].

In this chapter we introduce a shaped fitness function which combines difference evaluation functions with biasing the search with the best known set of collaborators, in order to address suboptimal solutions created by CCEAs. To test whether these problems are addressed adequately, two test domains are used. First, a *scatter domain*, which involves agents moving in a two dimensional plane in order to become as "spread out" as possible. Secondly, a *rover domain* involving robots gathering data from points of interest was utilized in order to test the algorithms

on a more real-world problem.

The contribution of this chapter is the development of a CCEA which:

- Shapes fitness functions using the difference evaluation function
- Biases search with optimal collaborators using a hall of fame approach, which is much less complex than approximating optimal collaborators in an ad-hoc manner
- Makes extremely efficient use of computational resources

In our experiments, this algorithm outperforms a CCEA using the system evaluation to assign fitness by an average of 48.7%, and is extremely computationally efficient. These results demonstrate not only the performance attainable by combining difference evaluations with other coordination mechanisms, but also that the computational cost of such an algorithm is low, allowing for use in real-world systems where agents have limited computational resources. The remainder of this chapter is organized as follows: Section 3.2 describes the domains analyzed. Section 3.3 describes the algorithms used in this research. Section 3.4 gives the experimental results. Finally, Section 3.5 discusses the results and concludes the chapter.

3.2 Evaluation Domains

In this section, we introduce the two problems analyzed in this work, and provide a detailed explanation of the system dynamics and evaluation functions used in

each domain.

3.2.1 Scatter Domain

The scatter domain used in this research is a variant of the mixing problem [110]. In the scatter domain, a team of agents on a two dimensional plane aim to move around and configure themselves to be as "spread out" as possible (Figure 3.1). The world is continuous, as are the actions of each agent. Each agent calculates the distance between itself and the closest teammate using the standard Euclidian distance. So, if there are N agents, each agent calculates how far away the closest agent is at any time t using:

$$\delta_i(t) = \min_j \left\{ \sqrt{(x_{i,t} - x_{j,t})^2 + (y_{i,t} - y_{j,t})^2} \middle| i \neq j \right\} \tag{3.1}$$

where $\{x_{i,t}, y_{i,t}\}$ is agent i's x and y position in the world at time t, and j is used to index all agents other than agent i. The total state of the system s_t is the set of all agent positions in the world at time t. The state of each agent is the relative x and y positions of the n closest agents. At each time step in an episode, an agent takes two actions Δ_x and Δ_y, corresponding to its x and y movements, respectively. The magnitude of these actions are bounded by some upper limit Δ_{max}, which requires that the agents take multiple actions over multiple time steps in order to traverse the domain. For our experiments, the agent's policies are represented by two layer feedforward neural networks with sigmoid activation functions, with

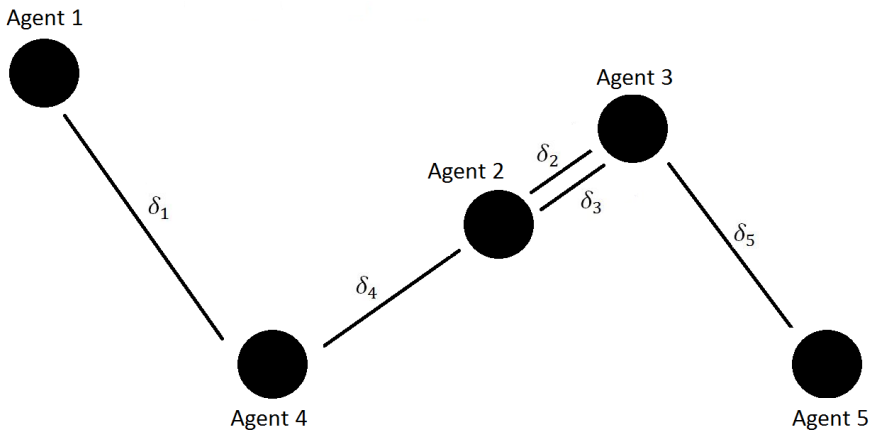

Figure 3.1: Scatter Domain Representation. Each agent i calculates the distance from itself to the closest agent as δ_i. The global evaluation function is the average of all of these distances. The goal in this domain is for the agents to be as "spread out" as possible.

outputs bounded between 0 and 1. The network inputs for each agent are that agent's local state (relative x and y position of the n closest agents), and the network outputs are mapped to agent motion in the world as follows:

$$\Delta_x = (2 \cdot o_1 - 1.0) \cdot \Delta_{max} \tag{3.2}$$

$$\Delta_y = (2 \cdot o_2 - 1.0) \cdot \Delta_{max} \tag{3.3}$$

where o_1 and o_2 are the neural network outputs. Thus, the distance agents may move in the x and y directions are bounded by $-\Delta_{max}$ and Δ_{max}. The system evaluation function for the scatter domain with N agents is the average minimum

squared distance between agents, given by:

$$G(s_t) = \frac{\sum_{i=1}^{N} \delta_i(t)}{N} \tag{3.4}$$

Thus, maximizing the average minimum distance between agents will result in maximizing the system evaluation.

3.2.2 Rover Domain

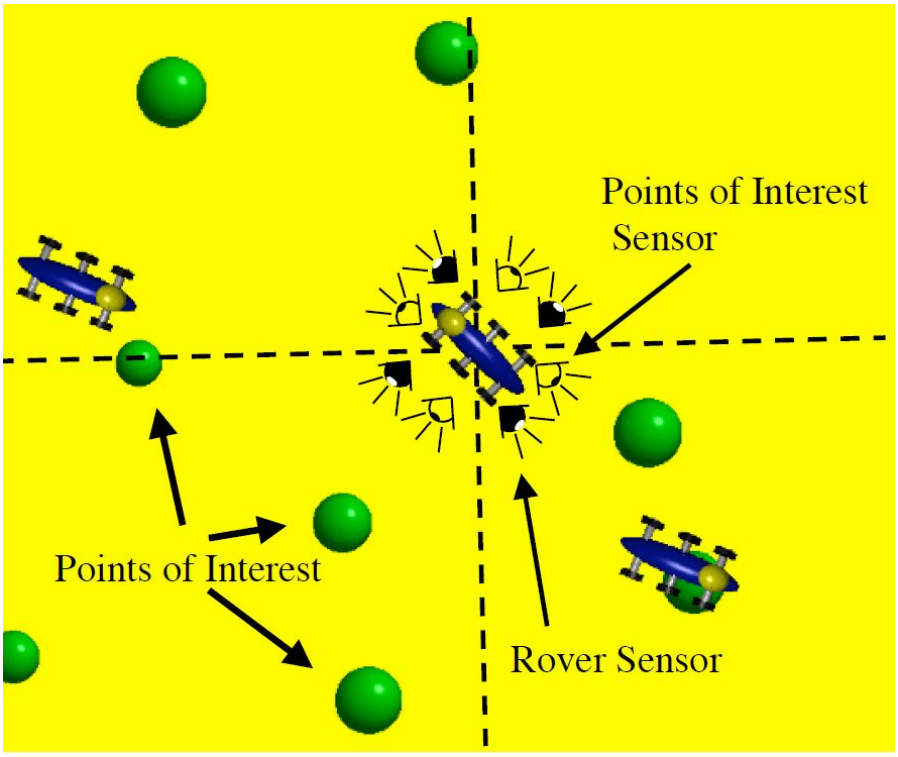

Figure 3.2: Rover Domain Representation. Each rover senses the closest rover and POI from each of its four sensing quadrants. The rovers must coordinate in order to effectively observe the POIs.

In the rover problem, a collective of rovers on a two dimensional plane aim to observe points of interest (POIs) scattered across the domain (Figure 3.2). 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:

$$\delta(x, y) = min\left\{||x - y||^2, \delta^2_{min}\right\} \tag{3.5}$$

The objective of the rovers is to maximize the observation values of the POIs over the course of an episode, and the system evaluation is calculated as:

$$G = \sum_t \sum_j \frac{V_j}{min_i \delta(L_j, L_{i,t})} \tag{3.6}$$

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

Each rover senses the world through eight sensors and map those inputs to two outputs representing the rover's motion in the x and y directions. The world is split into four quadrants relative to the rover's current orientation, with two sensors per quadrant. For each quadrant, the first sensor returns a function of the POIs in that quadrant. More specifically, the first sensor in quadrant q returns the sum of the values of the POIs in that quadrant divided by their squared distance

to the rover:

$$s_{1,q,i} = \sum_{j \in I_q} \frac{V_j}{\delta(L_j, L_i)} \tag{3.7}$$

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:

$$s_{2,q,i} = \sum_{i' in N_q} \frac{1}{\delta(L_{i'}, L_i)} \tag{3.8}$$

where N_q is the set of rovers in quadrant q. The eight sensors provide an approximate representation of the world based on the locations of POIs and other rovers. This representation reduces the location and number of rovers and POIs in each quadrant to an average number. It is of note that the value of each state variable is increased if there are many rovers/POIs in a quadrant, or if the rovers/POIs in a quadrant are close to the agent. The policies of each rover are represented by two layer feedforward neural networks with sigmoid activation functions. The inputs to the networks are the state variables defined in Equations 3.7 and 3.8. Each rover chooses actions which correspond to motion in the x and y directions, and the motions are bounded by a maximum distance Δ_{max}. The mapping from controller outputs to rover motion is the same as in the scatter domain, and is given in Equations 3.2 and 3.3. Although any rover may observe any POI, the system evaluation only takes into account the closest observation made for each POI. In this instantiation of the rover domain, the POI locations are static throughout each

experiment. The rovers must coordinate in order to achieve high POI coverage. An increasing system evaluation corresponds to better observation coverage of the POIs.

3.3 Algorithms

In this section we provide a detailed explanation of a standard cooperative co-evolutionary algorithm, as well as introduce three new CCEAs which implement difference evaluation functions:

1. Standard CCEA using system evaluation
2. CCEA using the difference evaluation
3. CCEA using lenient learners and the difference evaluation
4. CCEA using the hall of fame and the difference evaluation

The standard CCEA using the system evaluation is a "traditional" CCEA algorithm that we use a baseline to assess performance. The second and third algorithms are modifications of existing algorithms which address the problems of suboptimal convergence. The final algorithm is a modification of the hall of fame algorithm combined with the difference evaluation, which also aims to address suboptimal convergence and is the main contribution of this chapter.

3.3.1 Standard CCEA

In the standard CCEA, N coevolving populations of neural networks are utilized to form teams comprised of M agents. In the most general case, M is equal to N. One member of each population is extracted, and these agents are combined to form a team which operate in the problem domain. Each population is initially comprised of k neural networks, randomly initialized. At each generation, k successor networks are generated in each population, which are mutated versions of the parent networks. Then, $2k$ teams of M agents are formed by taking agents from each population and placing these agents into a team. The performance of each of the teams is then evaluated in the domain, and the fitness of every agent in the team is set according to the team's performance. Next, k networks from each population are selected to proceed to the next generation, with the fitness of a network influencing its selection probability. This process is repeated for a set number of generations. The standard CCEA is detailed in Algorithm 3.

In the standard CCEA, each member of a team receives equal credit for that team's performance. This form of credit assignment results in agents' fitness values to be heavily dependent upon the performance of teammates, because each member of the team receives equal credit for the team's performance. The standard CCEA is used as the baseline algorithmm, and serves as a comparison for the other algorithms considered.

Algorithm 3: Standard CCEA (See Section 3.4 for parameters)

```
Initialize N populations of k neural networks
foreach Generation do
    foreach Population do
        produce k successor solutions
        mutate successor solutions
    end
    for i = 1 → 2k do
        randomly select one agent from each population
        add agents to team T_i
        simulate T_i in domain
        assign fitness to each agent in T_i using G(s⃗, a⃗)
    end
    foreach Population do
        select k networks using ϵ-greedy
    end
end
```

3.3.2 CCEA with the Difference Evaluation

The CCEA with the difference evaluation is carried out in a similar manner to the standard CCEA, except that when a team of agents is evaluated, the fitness of each agent is calculated with the difference evaluation, rather than the system evaluation. Thus, the fitness of each agent of a team is calculated as that agent's contribution to the team's performance, rather than the value of the system evaluation function. The CCEA with the difference evaluation is equivalent to the CCEA detailed in Algorithm 3, except at the fitness assignment stage, the fitness of each agent is calculated with the difference evaluation rather than the system evaluation.

Thus, the key difference between the CCEA using the difference evaluation and

the standard CCEA is credit assignment for the agents. By utilizing the difference evaluation to assign fitness, the fitness of each agent becomes less dependent upon the actions of its teammates. Below, we derive the difference evaluation for the two domains used in this work.

Scatter Domain Directly computing the different evaluation using Equation 2.1 corresponds to applying Equation 2.1 to Equation 3.4. First, we define $\delta_i^j(t)$, which finds the distance from the closest agent to agent i, exluding agent j:

$$\delta_i^j(t) = \min_k \left\{ \sqrt{(x_{i,t} - x_{k,t})^2 + (y_{i,t} - y_{k,t})^2} | k \neq j \right\} \tag{3.9}$$

The difference evaluation function for the scatter domain is then defined as:

$$D_i(s_t) = \frac{\sum_{j=1}^{N} \delta_j(t)}{N} - \frac{\sum_{j \neq i} \delta_j^i(t)}{N} \tag{3.10}$$

However, this evaluation always increases the system evaluation, because of the nature of the system evaluation function. As the goal in this domain is to maximize average distance between agents, removing any agent will always have a positive effect on the system evaluation. As such agents will need to distinguish between very small positive variations, making the evaluation function in Equation 3.10 a poor choice for agent fitness function. Instead, in this work, we introduce a *default agent effect*. To compute agent i's fitness then, we replace agent i with this default

agent, which yields:

$$D_j = \frac{\sum_{j=1}^{N} \delta_j(t)}{N} - \frac{\left(\sum_{j \neq i} \delta_j^i(t)\right) + \delta_{def}(t)}{N} \tag{3.11}$$

where $\delta_{def}(t)$ is a distance associated with the default agent. This distance is set at the beginning of an experiment, and remains constant for each individual calculation of the difference evaluation. This default agent distance corresponds to the $c_{s,i}$ (*counterfactual*) term in Equation 2.1.

Rover Domain For the rover problem, the difference evaluation is calculated by directly applying Equation 2.1 to Equation 3.6:

$$D_i(L) = \sum_t \sum_j I_{j,i,t}(z) \left[\frac{V_j}{\delta(L_j, L_{i,t})} - \frac{V_j}{\delta(L_j, L_{k_j,t})} \right] \tag{3.12}$$

where k_j is the second closest rover to POI j, and $I_{j,i,t}(z)$ is an indicator function which returns 1.0 if and only if rover i is the closest rover to POI j at time t. If rover i is not the closest rover to any POI at time t, then its difference evaluation is zero, indicating that the rover is not contributing to the system evaluation function at time t.

3.3.3 CCEA with Lenient Learners and Difference Evaluation

The CCEA with lenient learners and the difference evaluation is carried out in a similar manner to the CCEA with the difference evaluation, except that each

agent is tested with multiple sets of collaborators (i.e. the agent will be placed in multiple teams), and the highest fitness achieved is the fitness assigned to that agent. As the algorithm progresses, agents become less lenient learners; that is, they are tested against fewer sets of collaborators. The CCEA with lenient learners and the difference evaluation is detailed in Algorithm 4.

It is important to note that in Algorithm 4, each member of each population is selected exactly m times, and the value of m is decreased as the algorithm progresses. This corresponds to agents being lenient in the early stages of evolution, and becoming less and less lenient as evolutionary time passes. Lenient difference evaluations shape fitness in two ways. First, the difference evaluations approximate the agent's individual contribution to the team's performance. Second, leniency ensures that suboptimal teammates do not negatively influence the agent's fitness.

3.3.4 CCEA with Hall of Fame and Difference Evaluation

As noted in Section 2, CCEAs have been shown to provide better solutions when the fitness of an individual is based partly on how it performs with its team, and partly on how it would perform if it were paired with optimal collaborators. However, estimating the fitness of an agent paired with optimal collaborators is a difficult task, especially in complex domains. Rather than estimating what optimal collaborators would be for a particular agent, the *hall of fame* method is altered to approximate the behavior of optimal collaborators.

The CCEA with the hall of fame and difference evaluation is carried out in

Algorithm 4: Lenient CCEA using Difference Evaluation (See Section 3.4 for parameters)

```
Initialize N populations of k neural networks
foreach Generation do
    foreach Population do
        produce k successor solutions
        mutate successor solutions
    end
    for i = 1 → 2k · m do
        randomly select one agent from each population
        add agents to team T_i
        simulate T_i in domain
        assign fitness to each agent in T_i using D(s⃗, a⃗)
        foreach Agent in team T_i do
            for l = 1 → m do
                add agent to randomly generated team T_i'
                simulate T_i' in domain
                calculate D(s⃗, a⃗) for the agent
                if D(s⃗, a⃗) > agent's fitness then
                    agent's fitness ← D(s⃗, a⃗)
                end
            end
        end
    end
    foreach Population do
        select k networks using ϵ-greedy
    end
end
```

a similar manner to the CCEA with the difference evaluation, except that the fitness assignment stage is altered. At the end of each generation, the team that achieves the highest system evaluation is compared against the hall of fame team. If that team achieved a higher system evaluation than the hall of fame team,

Algorithm 5: CCEA using Difference Evaluation and Hall of Fame (See Section 3.4 for parameters)

```
Initialize N populations of k neural networks
foreach Generation do
    foreach Population do
        produce k successor solutions
        mutate successor solutions
    end
    for i = 1 → 2k · m do
        randomly select one agent from each population
        add agents to team T_i
        simulate T_i in domain
        assign fitness to each agent with Eq. 3.14
    end
    foreach Team T_i do
        if G(s⃗, a⃗|T_i) > G(s⃗, a⃗|HOF) then
            assign T_i as hall of fame team
        end
    end
    foreach Population do
        select k networks using ϵ-greedy
    end
end
```

then it replaces the hall of fame team. When assigning fitness to each agent of a team, the difference evaluation of that agent is calculated, as well as the difference evaluation of that agent when it replaces an agent from the hall of fame team. The performance of the hall of fame team is nondecreasing with respect to evolutionary time, so this team approaches the optimal team as the CCEA progresses. The difference evaluation of an agent when compared with the hall of fame team is

calculated as:

$$D_{HOF,i} = G_{HOF+i} - G_{HOF} \tag{3.13}$$

where G_{HOF+i} is the system evaluation of the hall of fame team when agent i replaces the corresponding member of the hall of fame team, and G_{HOF} is the system evaluation of the best hall of fame team. So, in the CCEA with the hall of fame and difference evaluation, the fitness of an agent is calculated as:

$$F(i) = \alpha \cdot D_i + (1 - \alpha) \cdot D_{HOF,i} \tag{3.14}$$

where D_i is the difference evaluation of agent i when collaborating with its team, $D_{HOF,i}$ is the agent's difference evaluation when paired with the best hall of fame team as in Equation 3.13, and $\alpha \in [0, 1]$ is a weight corresponding to the relative importance of the difference evaluation and the difference evaluation with estimated optimal collaborators. The CCEA with the hall of fame and difference evaluation is detailed in Algorithm 5.

By assuming that the hall of fame team is the set of optimal collaborators for any agent, the complexities of estimating what a set of optimal collaborators would be are eliminated. The CCEA using the difference evaluation and the hall of fame includes shaped fitness functions to tell agents what their individual contributions to team performance are, as well as biasing the fitness functions using the concept of estimated optimal collaborators via the hall of fame. This approach modifies the hall of fame algorithm in two ways. First, the hall of fame is now comprised of *teams*, rather than *individuals*. Secondly, the hall of fame is now utilized in

cooperative coevolution, rather than *competitive* coevolution.

3.3.5 Computational Complexity Analysis

It is important to note the differences in computational complexity of each of the four algorithms analyzed. Each algorithm will produce a different level of performance, indicating the effectiveness of each fitness assignment operator used. However, each of these fitness assignment operators have varying compuational costs, so we must analyze not only converged performance levels, but performance as a function of computational cost. We assume that each team is comprised of N agents. For the standard CCEA using the system evaluation function as a fitness assignment operator, one call to $G(\vec{s}, \vec{a})$ is required to assign fitness to each member of that team. For the CCEA using difference evaluations, $N + 1$ calls to $G(\vec{s}, \vec{a})$ are required to assign fitness to each agent in the team. The value of $G(\vec{s}, \vec{a})$ is calculated for the entire team, and then $G(\vec{s}, \vec{a})$ must be called N times to find the value of $G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i})$ for each agent in order to calculate the difference evaluation function. For the CCEA using lenient learners and the difference evaluation, $(M + 1)N + 1$ calls to $G(\vec{s}, \vec{a})$ are required to assign fitness to each agent, where M is the number of teams each agent is tested with. The difference evaluation is calculated for each agent on the team, requiring $N + 1$ calls to $G(\vec{s}, \vec{a})$. Then, each agent is placed on M different teams and the difference evaluation for that agent is calculated for each team tested, requiring $2M$ calls to $G(\vec{s}, \vec{a})$ per agent. For the CCEA using the hall of fame and difference evaluation,

$2N + 1$ calls to $G(\vec{s}, \vec{a})$ are required to assign fitness to each agent. The difference utillity is calculated for each agent in the team, requiring $N + 1$ calls to $G(z)$. Then, the difference evaluation for each agent when placed on the hall of fame team must be calculated, requiring N more calls to $G(\vec{s}, \vec{a})$. The computational cost of each fitness assignment operator is summarized in Table 3.1.

Fitness Assignment Operator	Calls to $G(\vec{s}, \vec{a})$ Required per Team
$G(\vec{s}, \vec{a})$	1
$D_i(\vec{s}, \vec{a})$	$N + 1$
Lenient $D_i(\vec{s}, \vec{a})$	$(M + 1)N + 1$
Hall of Fame with $D_i(\vec{s}, \vec{a})$	$2N + 1$

Table 3.1: Computational cost to assign fitness to each member of a team for each fitness assignment operator tested, where N is the number of agents per team and M is the leniency value

As seen in Table 3.1, the computational cost of each of the fitness assignment operators considered varies widely, from 1 call to $(M + 1)N + 1$ calls of the system evaluation function required per team. Although more computationally expensive algorithms may provide better converged performance, it is important to note the computational complexity of each algorithm, especially if the time allocated for learning is limited. Thus, we will analyze not only performance as a function of evolutionary time, but performance as a function of computational cost.

3.4 Experimental Results

The algorithms outlined in Section 3.3 were all tested in the scatter domain and the rover domain, with team sizes varying from 10 to 100 agents. For experiments

with 10 agent teams, 10 coevolving populations of 200 members each were used. For experiments with 100 agent teams, 100 coevolving populations of 25 members each were utilized. For the standard CCEA, Equation 3.4 with a default agent distance of 1.0 was used to assign fitness in the scatter domain and Equation 3.6 was used in the rover domain. For the CCEA algorithm with the difference evaluation, Equation 3.11 was used to assign fitness for the scatter domain and Equation 3.12 for the rover domain. For the CCEA with lenient learners and the difference evaluation, the same fitness equations were used as in the CCEA with the difference evaluation experiments. The leniency value m is set to 10, meaning that each agent is assigned fitness based on its best performance from 10 different sets of collaborators. The CCEA with the hall of fame and difference evaluation had the same fitness assignments as in the CCEA with the difference evaluation. When calculating the fitness from equation 3.13, α was initially set to 1, and linearly decreased to 0.25 at the end of the experiment. The value of α initially is large, which results in the difference evaluation when an agent on the hall of fame team is initially small. This is because at the beginning of the experiment, the hall of fame team is severely suboptimal. As evolutionary time progresses, the hall of fame team improves in performance, and the value of the evaluation associated with the hall of fame team plays a larger role in fitness assignment. For all experiments, neural network controllers were initialized with random weights drawn from a Gaussian distribution with zero mean and unit variance. Network mutation was carried out by adding values drawn from a Gaussian distribution to a set of randomly selected network weights. In the beginning of the evolution, one

weight per network was mutated with a standard deviation of 1.0. At the end of the evolution, each network weight was mutated with a standard deviation of 0.1. These mutation parameters were varied linearly throughout evolution. For each experiment, 100 statistical runs were completed, with the standard error in the mean ($\sigma/\sqrt{N}$) being reported as error bars. The experiment details and results are given in the following sections.

3.4.1 Scatter Domain

First, we applied each of the four CCEA algorithms to the scatter domain. For the 10 agent experiment, the world was set to a 10 by 10 plane world and run for 10 time steps, and Δ_{max} was set to 1.0. For the 100 agent experiment, the world was set to a 31.6 by 31.6 plane world and run for 10 time steps, and Δ_{max} was set to 3.16. These values were chosen such that the plane area to number of agents ratio was constant for each experiment, and the agents could traverse the world in the same number of time steps. At the beginning of each experiment, the agents all started at the center of the plane worlds. Figure 3.3 shows the learning curve for the 10 agent problem. Figure 3.4 shows the learning curve for the 100 agent problem. Finally, Figure 3.5 shows the scaling properties of each algorithm in the scatter domain.

All three algorithms using the different evaluation function outperformed the standard CCEA in this domain. This is not surprising, because credit assignment in this algorithm is highly subjective, and the fitness of each agent was greatly

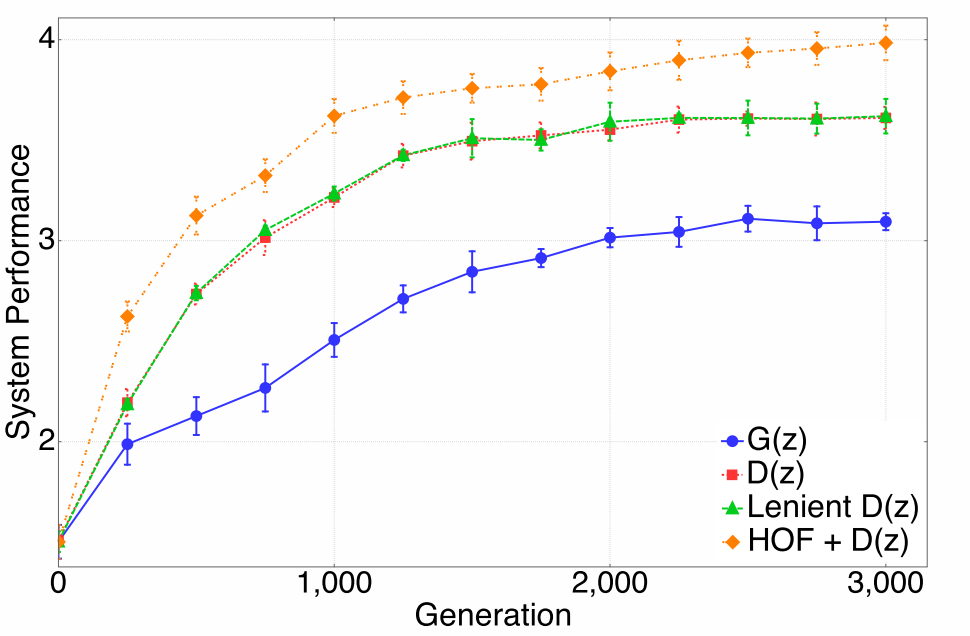

Figure 3.3: Performance of each algorithm in the scatter domain, with 10 agents. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods tested.

influenced by its teammates. The CCEA with the difference evaluation and the CCEA with leniency and the difference evaluation performed almost identically in the 10 agent case, but the addition of leniency provided improved performance in the 100 agent case. This is an interesting result, and gives insight to the properties of leniency. Leniency and the difference evaluation both perform similar functions. Leniency aims to reduce the subjectiveness of credit assignments in CCEAs by partnering agents with multiple sets of collaborators. By testing an agent with multiple teams and taking the highest fitness achieved, the likelihood that an agent's fitness is too strongly biased by its teammates is minimized. The difference evaluation also reduces the subjectiveness of credit assignment, by iso-

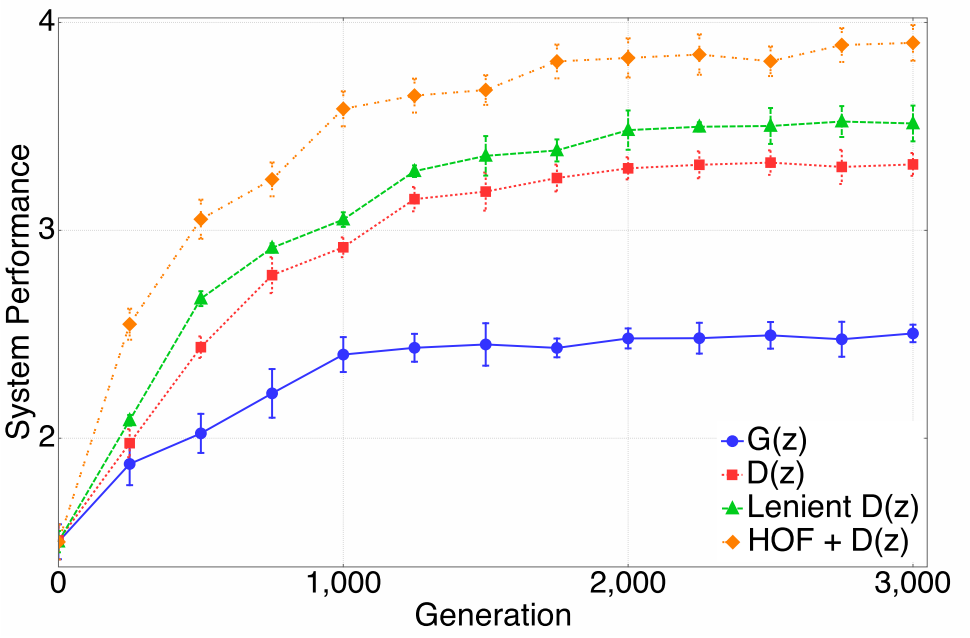

Figure 3.4: Performance of each algorithm in the scatter domain, with 100 agents. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods tested.

lating an agent's individual contribution to its team's performance. Thus, leniency and the difference evaluation achieve a similar goal, although in different manners. However, as the problem becomes more complex by increasing the team size, coupling both approaches provides benefits, as seen in Figure 3.5. As the team size is increased, the performance of the CCEA with the difference evaluation and lenient learners outperforms the CCEA with the difference reward by larger and larger margins.

The CCEA with the hall of fame and difference evaluations performed the best out of all of the algorithms tested. As noted in Chapter 2, one of the key properties of leniency is that an agent will not receive a low fitness evaluation simply because

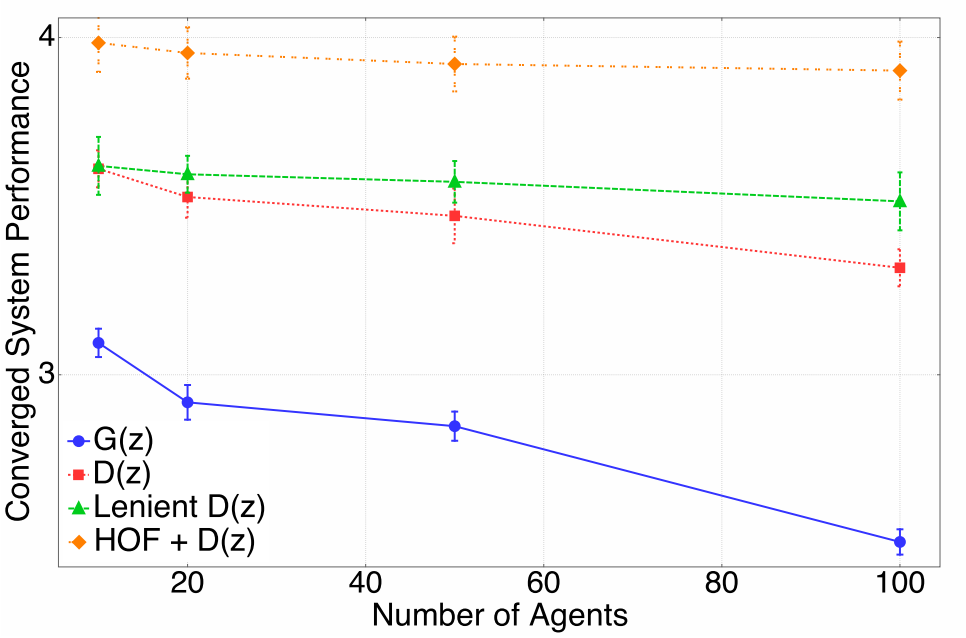

Figure 3.5: Scaling Performance in Scatter Domain. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods tested, and this difference in performance increases with team size. The addition of leniency does not significantly help when there are a small number of agents, but as the team size goes up, leniency becomes increasingly useful.

it is paired with suboptimal collaborators. With the hall of fame approach, this advantage is also present, as the hall of fame team is known to provide good system performance. In this manner, the hall of fame ensures agents do not receive poor fitness assignment values as a result of suboptimal teammates, while requiring far less computational expense than leniency. Both leniency and the hall of fame approach bias the search towards finding agents which perform well with a set of high-performing collaborators. However, as the hall of fame approach utilizes the best team known at that point in the algorithm, the searched is biased in a more favorable area of the search space, which results in the hall of fame with difference

evaluations outperforming leniency with difference evaluations.

Although the hall of fame and leniency have similar effects on agent learning, there are two key differences. First, the hall of fame approach is much less computationally expensive, as an agent only needs to be tested with one set of collaborators, rather than many. Second, both approaches bias the search towards assigning fitness values based on optimal collaborators, but the hall of fame approach biases this search with the best known team, as opposed to leniency biasing the search with randomly sampled teams. In this manner, the hall of fame approach results in a more directed and precise biasing of the coevolutionary search algorithm.

3.4.2 Rover Domain

Next, we applied each of the four CCEA algorithms to the rover domain. At the beginning of each experiment, n POIs were placed randomly in the domain, and their positions remained constant throughout each experiment, where n is equivalent to the number of agents in the domain. The minimum observation distance δ_{min} was set to 0.1. The simulations were carried out in a 10 by 10 world for 25 time steps, and Δ_{max} was set to 1.0. At the beginning of each simulation, each rover started in the center of the world. Each algorithm was tested over 50 statistical runs. Figure 3.6 shows the results for 10 agent teams, Figure 3.7 shows the results for 100 agent teams, and Figure 3.8 shows the scaling results for the rover domain.

As in the scatter domain experiments, all three algorithms using the different

evaluation function outperformed the standard CCEA in the rover domain. This can be attributed to the fact that the system evaluation does not give an agent good feedback on its individual contribution to the team's performance, and the agent thus has a difficult time learning an optimal policy. In the scatter domain, the CCEA with the difference evaluation and the CCEA with lenient learners and the difference evaluation performed nearly identically with 10 agent teams, but leniency became more important as the team size went up. In the more complex rover domain, the CCEA with lenient learners and the difference evaluation performed slightly better than the CCEA with the difference reward, and this performance gain also increased with the team size. This indicates that although leniency and the difference evaluation have similar effects on the learning process, leniency may become more beneficial in CCEAs as the domain becomes more complex or the number of cooperating agents increases.

As in the scatter domain, the CCEA with the hall of fame and difference evaluation performed the best out of all algorithms tested. This further supports the conclusion that biasing the CCEA with hall of fame teams approximating optimal collaborators, as well as shaping the fitness functions, helps guide the search towards better solutions. As seen in Figure 3.8, the difference in performance between the CCEA with the difference evaluation and hall of fame and the CCEA with the global evaluation function increases with team size, indicating that for increasingly complex systems, this new algorithm becomes more and more useful. In all the experiments that were conducted, the CCEA with the hall of fame and difference evaluation significantly outperformed all other algorithms, and was able

to easily estimate optimal collaborators in order to bias the fitness. This fitness biasing, in addition to shaping the fitness values with the difference evaluation, contributed to significantly better performance than any other algorithm tested.

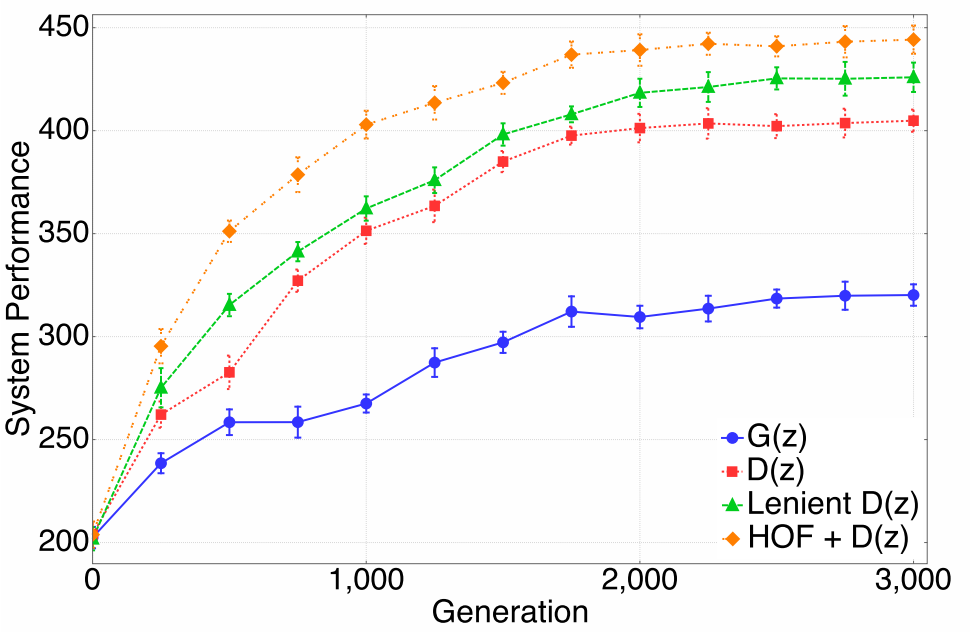

Figure 3.6: Performance of each algorithm in the rover domain, with 10 agents and 10 POIs. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods tested.

3.4.3 Computational Cost Analysis

Sections 3.4.1 and 3.4.2 demonstrated that combining difference evaluations with either leniency or hall of fame methods can significantly improve converged sys-

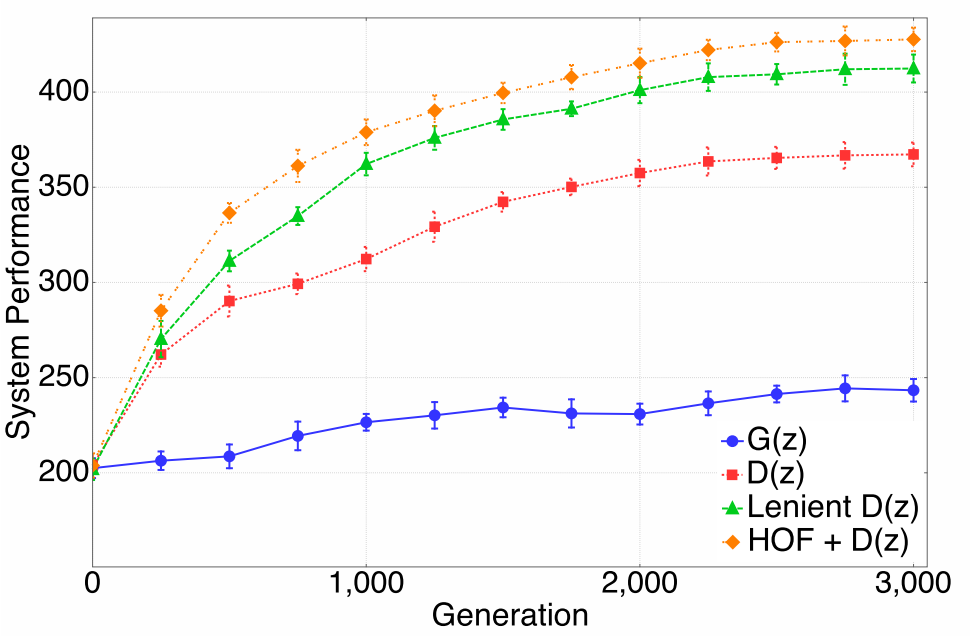

Figure 3.7: Performance of each algorithm in the rover domain, with 100 agents. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods tested.

tem performance. However, as noted in Section 3.3.5, the computational cost of each of the four algorithms varies greatly. We now analyze the performance of each algorithm as a function of computational cost for each of the experiments conducted.

Scatter Domain

The most expensive fitness assignment operator utilized in this work is the lenient difference evaluation, requiring $(M+1)N+1$ calls to $G(\vec{s},\vec{a})$ in order to assign fitness to each member of a team. In the same number of generations, the lenient

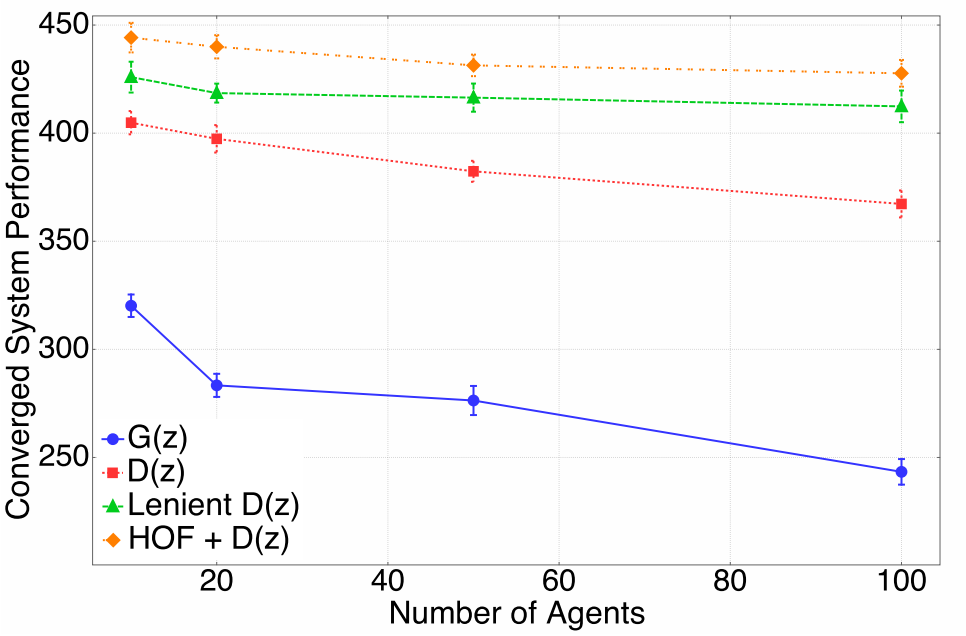

Figure 3.8: Scaling Performance in Rover Domain. The CCEA with the difference evaluation and hall of fame biasing outperforms all other methods, and the gap in performance increases with team size.

difference evaluation calls $G(\vec{s}, \vec{a})$ many more times than the other fitness assignment operators, resulting in much more information available to use to provide feedback to learning agents. It is therefore important to analyze performance as a function of computational cost, in order to compare performance of each fitness assignment operator when given the same amount of computational resources. The performance in the 10 agent scatter domain as a function of calls to $G(\vec{s}, \vec{a})$ is given in Figure 3.9.

As seen in Figure 3.9, although difference evaluations and lenient difference evaluations perform nearly identically with respect to evolutionary time, difference evaluations perform significantly better with respect to computational cost.

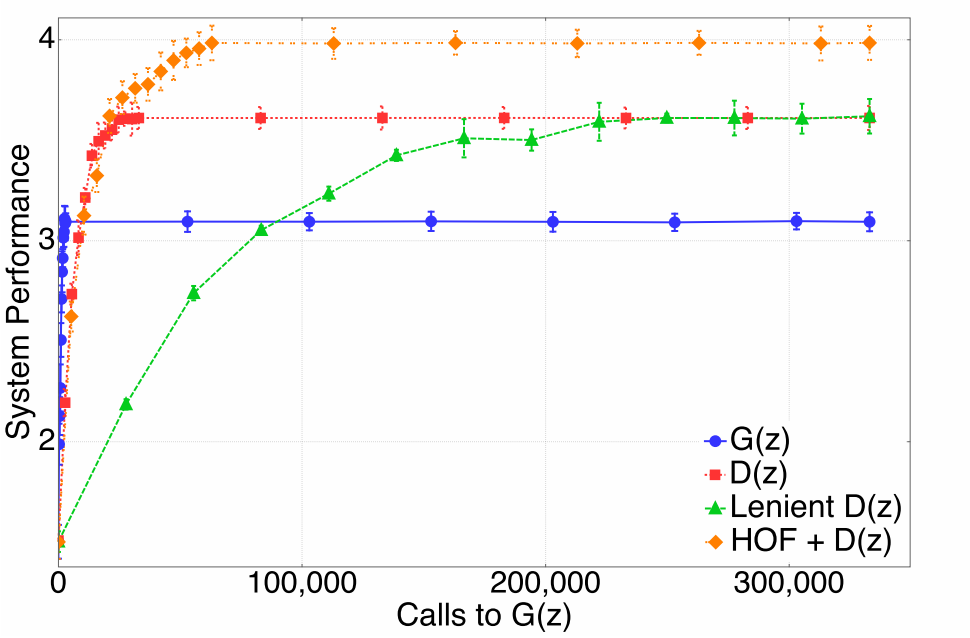

Figure 3.9: Performance as a function of computational cost of each algorithm in the scatter domain, with 10 agents.

Using difference evaluations results in about ten times faster convergence than lenient difference evaluations. So, although the results in Section 3.4.1 indicate that difference evaluations and lenient difference evaluations perform nearly identically in some situations, difference evaluations are much more computationally efficient. Another interesting result seen in Figure 3.9 is that combining the hall of fame and difference evaluations is extremely efficient. Combining the hall of fame and difference evaluations converges five times faster than lenient difference evaluations, and provides superior converged performance. Thus, combining hall of fame and difference evaluations provides the best converged performance and is computationally efficient.

The performance in the 100 agent scatter domain as a function of calls to $G(\vec{s}, \vec{a})$ is given in Figure 3.10. As in the 10 agent case, difference evaluations are much more efficient than lenient difference evaluations. However, lenient difference evaluations do provide better converged performance. The key result seen in Figure 3.10 is that combining the hall of fame and difference evaluations results in the best converged performance, and this fitness assignment operator makes extremely efficient use of computational resources.

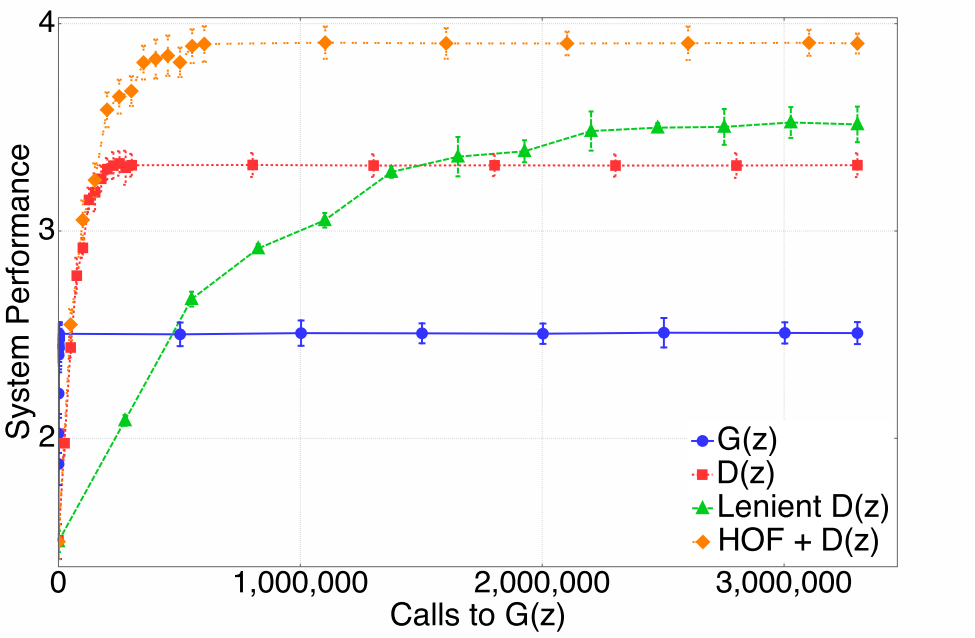

Figure 3.10: Performance as a function of computational cost of each algorithm in the scatter domain, with 100 agents.

Rover Domain

The computational cost analysis of each fitness assignment operator in the rover domain yields similar results to that of the scatter domain. The performance levels in the rover domain with 10 and 100 agents as a function of computational cost are shown in Figure 3.11 and 3.12, respectively.

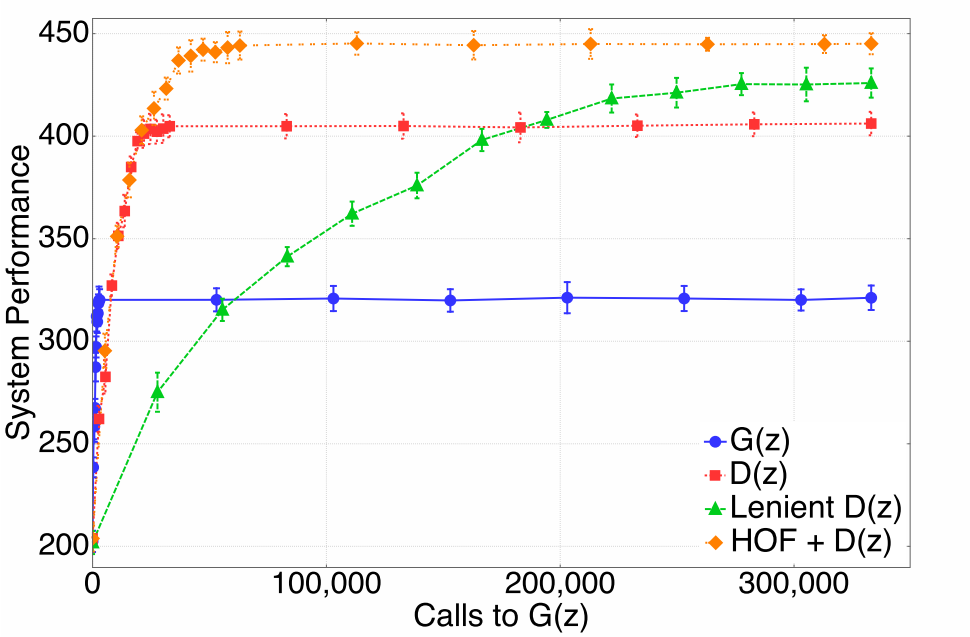

Figure 3.11: Performance as a function of computational cost of each algorithm in the rover domain, with 10 agents.

In the more complex rover domain, lenient difference evaluations always converge to better performance than difference evaluations, as seen in Figures 3.11 and 3.12. However, difference evaluations converge around ten times faster. As in the scatter domain, combining difference evaluations with hall of fame methods results in superior converged performance with much less computational cost than

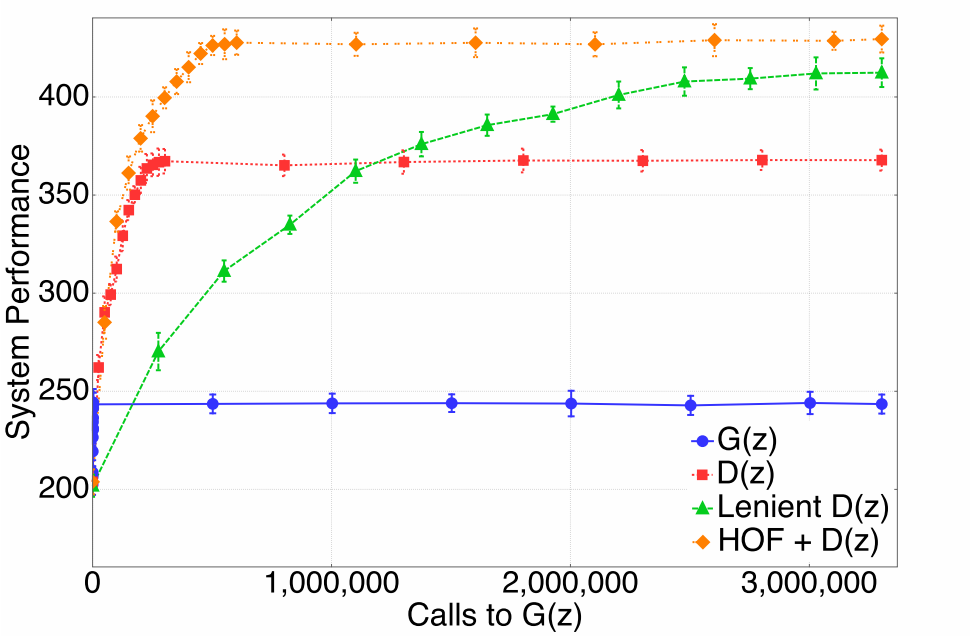

Figure 3.12: Performance as a function of computational cost of each algorithm in the rover domain, with 100 agents.

lenient difference evaluations, and about the same computational cost as standard difference evaluations.

A few conclusions regarding performance can be made from the information presented in Figures 3.9-3.12. First, although lenient difference evaluations converge to better performance levels than difference evaluations in complex domains, difference evaluations are far more computationally efficient. Second, and more importantly, combining the hall of fame with difference evaluations results in the best converged performance, and is extremely computationally efficient. This means that regardless of the time allocated for learning, the best fitness assignment operator analyzed is the combination of the hall of fame and difference evaluations. If

enough time is allocated for the algorithm to converge, then the best performance is attained. If the time available for learning is limited, combining the hall of fame and difference evaluations might not result in convergence, but will significantly outperform lenient difference evaluations and will slighlty outperform standard difference evaluations. Thus, not only does the combination of the hall of fame and difference evaluations result in the best converged performance, but it makes the most efficient use of each call to the system evaluation function $G(\vec{s}, \vec{a})$; this demonstrates that this fitness assignment operator is far superior to all operators examined.

3.5 Summary

This chapter presented three CCEA algorithms where leniency and hall of fame methods were used in combination with fitness shaping. Combining hall of fame and difference evaluation outperformed all other algorithms in two different domains. It is known that shaping fitness functions can greatly improve the efficacy of a CCEA [64], but this often isn't enough to obtain optimal performance. It has also been shown that biasing a CCEA with an estimate optimal collaborators results in a more effective search for optimal policies, but this estimate is problematic because it is an ad hoc, domain dependent estimate. Our algorithm circumvents the problem of estimating optimal collaborators, so the CCEA search is easily biased in a domain independent fashion. Furthermore, the algorithm shapes agent fitnesses in order to provide each agent a measure of its individual contribution to

the system objective.

A particularly interesting result was that in the simpler scatter domain, using the difference evaluation to shape the fitness functions performed equivalently to a method which used lenient learners and the difference evaluation for 10 agent teams. Intuitively, the difference evaluation and lenient learners both aim to achieve the same goal, which is to isolate an agent's individual contribution to the system evaluation. This is one key reason their performance was similar in the simpler domains. However, with larger teams or a more complicated domain, leniency in addition to the difference evaluation performed better than the difference evaluation alone.

Although the combination of the hall of fame and difference evaluations converged to the highest performance levels, it is important to analyze the computational cost of fitness assignment operators in cooperative coevolutionary algorithms. We find that not only does this combination result in the highest converged performance, but it makes the most efficient use of computational resources out of all the fitness assignment operators tested. The computational cost analysis clearly demonstrates that the combination of the hall of fame and difference evaluation functions is the best fitness assignment operator out of the group analyzed in this research.

As noted in Chapter 1, the first key research question regarding difference evaluations involved determining the compatibility of difference evaluations with other coordination mechanism, and the performance attainable by these combinations. In this chapter, we demonstrated that difference evaluations are compatible

with other coordination mechanisms, and when combined with leniency or a hall of fame approach can significantly improve system performance as compared to any of these operators acting alone. Further, not only does combining difference evaluations with the hall of fame result in significant performance gains, it is more computationally efficient than any of these operators acting alone. The results from this chapter demonstrate that significant performance gains are attainable when difference evaluations are incorporated into agent fitness functions, and address the first research question regarding difference evaluation functions.

Chapter 4 – An Evolutionary Game Theoretic Perspective on Difference Evaluation Functions

4.1 Motivation

Coordinating multiple agents to achieve a system-level objective is a critical area of research, and is essential for many applications including air traffic control and mobile robot coordination [5, 39]. One approach to solving this coordination problem is to use Cooperative Coevolutionary Algorithms (CCEAs), which evolve multiple populations simultaneously and evaluate fitness of inidivuals based on their interactions with other agents in the system [90]. In CCEAs, agents' fitness assignments are influenced by agents from other populations. Thus, CCEAs tend to create agents which perform well with many types of agents, rather than specializing to perform optimally with a specific set of collaborators. This results in CCEAs typically producing stable, suboptimal, solutions [19, 49, 90]. In order for CCEAs to provide better coordination in multiagent systems, fitness functions must be shaped in order to provide better agent-specific feedback.

This chapter is focused specifically on the theoretical guarantees regarding difference evaluation functions as fitness assignment operators in CCEAs, using an evolutionary game theoretic setting [17, 24, 53, 78, 100, 103, 108]. Difference evaluations approximate an individual agent's impact on the system evaluation function,

and assigns a fitness to that agent based on the effectiveness of its particular policy. The difference evaluation has produced excellent empirical results when used as a fitness assignment operator in CCEAs [39], as seen in Chapter 3. However, no prescriptive theoretical analysis of the advantages of difference evaluation functions has been conducted, indicating when their use is beneficial. In this chapter, we derive conditions under which difference evaluations are expected to improve performance in CCEAs, using evolutionary game theoretic methods.

The contributions of this chapter are to:

- Theoretically derive conditions under which the difference evaluation function increases the expected payoff for optimal actions corresponding to optimal Nash equilibrium points.
- Empirically demonstrate the effects of these conditions being met, and show the empirical results are consistent with the theoretical analysis of difference evaluations.

The remainder of this chapter is organized as follows: Section 4.2 presents the evolutionary game-theoretic model for cooperative coevolutionary algorithms. Section 4.3 demonstrates how difference evaluation functions are incorporated into the evolutionary game theoretic model. Section 4.4 derives conditions under which the difference evaluation function increases the expected payoff of optimal actions. Section 4.5 gives empirical results in multiple games which support the theoretical results. Section 4.6 extends the theoretical results to systems with more than two agents. Finally, Section 4.7 concludes the chapter.

4.2 EGT Model for Cooperative Coevolution

Our analysis is restricted to cooperative coevolutionary algorithms with two agents learning in stateless domains, where each agent has a finite number of actions. The model assumes that the populations are infinite, and that the proportions of individuals in the populations are computed at each time step during evolution. If the first agent has a finite number of n distinct actions it can take, then its population at each generation is an element of $\Delta^n = \{p \in [0,1]^n | \sum_{i=1}^{n} p_i = 1\}$. A higher value x_i corresponds to a higher probability that the agent selects action i. If the second agent has m actions to choose from, then its population at each generation is an element of $\Delta^m = \{q \in [0,1]^m | \sum_{i=1}^{m} q_i = 1\}$. Assuming a symmetric system where both agents are equally rewarded, then the payoff matrix C is used to compute the fitnesses in one population, and C^T is used to calculate fitnesses for the other population. The EGT model for CCEAs is defined as in [91, 122]:

$$u_i^{(t)} = \sum_{j=1}^{m} c_{ij} y_j^{(t)} \tag{4.1}$$

$$w_j^{(t)} = \sum_{i=1}^{n} c_{ij} x_i^{(t)} \tag{4.2}$$

$$x_i^{(t+1)} = \left(\frac{u_i^{(t)}}{\sum_{k=1}^{n} x_k^{(t)} u_k^{(t)}} \right) x_i^{(t)} \tag{4.3}$$

$$y_j^{(t+1)} = \left(\frac{w_j^{(t)}}{\sum_{k=1}^{m} y_k^{(t)} w_k^{(t)}} \right) y_j^{(t)} \tag{4.4}$$

where $x^{(t)}$ and $y^{(t)}$ represent the proportions of genotypes (actions) in the two

populations at generation t, and $x^{(t+1)}$ and $y^{(t+1)}$ represent the proportions at the next generation. Equations 4.1 and 4.2 compute the fitness of each action in the two populations. The fitness of action i is estimated as the mean payoff over pairwise collaborations with every action in the other population. Equations 4.3 and 4.4 calculate the distributions of the two populations for the next generation. Equations 4.1-4.4 define how the populations in a CCEA progress as evolutionary time progresses.

4.3 EGT Model with Difference Evaluation Functions

In the following sections, we define payoff matrices which utilize the difference evaluation function, and incorporate difference evaluations into the EGT model for CCEAs (Section 4.2).

4.3.1 Difference Payoff Matrices

Consider the case where we have a global payoff matrix C, and we want D^1 and D^2 to be payoff matrices defined by the difference evaluation function (Section 2.3) for each agent. Now, the game is not equal payoff, meaning that each agent receives a different individual payoff (from the difference evaluation) based on the joint action taken. We assume that the counterfactual term in the difference evaluation function is the average payoff for an agent across all actions, given the action of

the collaborating agent. Thus we have:

$$d_{ij}^1 = c_{ij} - \frac{\sum_{k=1}^{n} c_{kj}}{n} + \alpha \tag{4.5}$$

$$d_{ij}^2 = c_{ij} - \frac{\sum_{k=1}^{m} c_{ik}}{m} + \alpha \tag{4.6}$$

The payoff matrices defined by Equations 4.5 and 4.6 are termed the *difference payoff matrices*. Note that α is a constant to ensure that at least one payoff matrix element is non-negative. Further, note that no hand tuning of the agent-specific payoff matrices is performed. These are simply the payoff matrices directly defined by the difference evaluation function.

4.3.2 EGT Model

The EGT model for CCEAs when using the difference evaluation function is:

$$u_i^{(t)} = \sum_{j=1}^{m} d_{ij}^1 y_j^{(t)} \tag{4.7}$$

$$w_j^{(t)} = \sum_{i=1}^{n} d_{ij}^2 x_i^{(t)} \tag{4.8}$$

$$x_i^{(t+1)} = \left(\frac{u_i^{(t)}}{\sum_{k=1}^{n} x_k^{(t)} u_k^{(t)}} \right) x_i^{(t)} \tag{4.9}$$

$$y_j^{(t+1)} = \left(\frac{w_j^{(t)}}{\sum_{k=1}^{m} y_k^{(t)} w_k^{(t)}} \right) y_j^{(t)} \tag{4.10}$$

Equations 4.7 and 4.8 can be rewritten as the following for the difference payoff matrices:

$$u_i^{(t)} = \sum_{j=1}^{m} \left(c_{ij} - \frac{\sum_{k=1}^{n} c_{kj}}{n} + \alpha \right) y_j^{(t)} \tag{4.11}$$

$$w_i^{(t)} = \sum_{i=1}^{m} \left(\frac{\sum_{k=1}^{m} c_{ik}}{m} + \alpha \right) x_j^{(t)} \tag{4.12}$$

The EGT model for CCEAs using the difference evaluation function is identical to the standard EGT model, except that the fitness of each agent is assigned using the difference payoff matrices D^1 and D^2 rather than the system payoff matrix C.

4.4 Expected Payoffs Theory

In this section, we present the results of a theoretical analysis on the effectiveness of difference payoff matrices and global payoff matrices, using an evolutionary game-theoretic setting. For this analysis, we make the following assumptions:

Assumption I: All elements of the payoff matrix are positive.

Assumption II: Each payoff matrix has at least two distinct elements (each element of the payoff matrix does not have the same value).

If we encounter a payoff matrix with all negative values, then we add a constant term to each element in the payoff matrix in order to ensure that the payoff matrix elements are positive. We assume the payoff matrix has at least two distinct elements so there exists at least one joint action which yields a higher payoff than

some other joint action, which is necessary for the proof. It is of note that if this assumption does not hold, then every element of the payoff matrix has the same value, meaning that no matter what the joint action of the agents is, the payoff will remain the same. So although this type of game has been eliminated from our analysis, it is of no theoretical interest.

4.4.1 Global Payoff Matrix

The expected payoff for the first agent taking action i is:

$$E[C^1(i)] = \frac{1}{m}\sum_{j=1}^{m} c_{ij} \tag{4.13}$$

where C^1 is the payoff matrix for the first agent. As we are comparing performance of the CCEA using either the system payoff matrix or difference payoff matrices, we must normalize the payoffs in order to have a fair comparison. The normalized payoff for the first agent taking the optimal action $\bar{E}[C^1(i^*)]$ is the expected payoff of the optimal action divided by the sum of expected payoffs for all possible actions:

$$\begin{aligned}
\bar{E}[C^1(i^*)] &= \frac{E[C^1(i^*)]}{\sum_{i=1}^{n} E[C^1(i)]} \\
&= \frac{E[C^1(i^*)]}{\sum_{i=1}^{n}\left[\frac{1}{m}\sum_{j=1}^{m} c_{ij}\right]} \\
&= \frac{E[C^1(i^*)]}{n \cdot c_{avg}}
\end{aligned} \tag{4.14}$$

Where c_{avg} is the average payoff of the global payoff matrix, and is defined as:

$$c_{avg} = \frac{1}{m \cdot n} \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} \tag{4.15}$$

The expected global payoff for the second agent taking action j is:

$$E[C^2(j)] = \frac{1}{n} \sum_{i=1}^{n} c_{ij} \tag{4.16}$$

where C^2 is the payoff matrix for the second agent. Note that $C^1 = C^2$ in this case, as both agents use the system payoff matrix. This notation is introduced for clarity when difference payoff matrices are utilized, and each agent uses a different payoff matrix. The normalized expected payoff for the second agent taking the optimal action $\bar{E}[C^2(j^*)]$ is defined as:

$$\bar{E}[C^2(j^*)] = \frac{E[C^2(j^*)]}{\sum_{j=1}^{m} E[C^2(j)]} \tag{4.17}$$

$$= \frac{E[C^2(j^*)]}{\sum_{j=1}^{m} \left[\frac{1}{n} \sum_{i=1}^{n} c_{ij}\right]} \tag{4.18}$$

$$= \frac{E[C^2(j^*)]}{m \cdot c_{avg}} \tag{4.19}$$

4.4.2 Difference Payoff Matrix

The expected payoff for the first agent when using the difference payoff matrix and taking action i is:

$$\begin{aligned} E[D^1(i)] &= \frac{1}{m}\sum_{j=1}^{m} d_{ij}^1 \\ &= \frac{1}{m}\sum_{j=1}^{m}\left(c_{ij} - \frac{1}{n}\sum_{k=1}^{n} c_{kj} + \alpha\right) \\ &= E[C^1(i)] - c_{avg} + \alpha \end{aligned} \tag{4.20}$$

The normalized expected payoff for the first agent when taking the optimal action and using difference payoff matrices is defined as:

$$\begin{aligned} \bar{E}[D^1(i^*)] &= \frac{E[D^1(i^*)]}{\sum_{i=1}^{n} E[D^1(i)]} \\ &= \frac{E[C^1(i^*)] - c_{avg} + \alpha}{\sum_{i=1}^{n}\left(E[C^1(i)] - c_{avg} + \alpha\right)} \\ &= \frac{E[C^1(i^*)] - c_{avg} + \alpha}{\sum_{i=1}^{n}\left(E[C^1(i)]\right) - n \cdot c_{avg} + n \cdot \alpha} \\ &= \frac{E[C^1(i^*)] - c_{avg} + \alpha}{n \cdot \alpha} \end{aligned} \tag{4.21}$$

Setting $\alpha = c_{max}$ ensures that the values in the difference payoff matrix are positive, so we have:

$$\bar{E}[D^1(i^*)] = \frac{E[C^1(i^*)] - c_{avg} + c_{max}}{n \cdot c_{max}} \tag{4.22}$$

where c_{max} is defined as:

$$c_{max} = max\left\{c_{ij} | i \in [1, n], j \in [1, m]\right\} \tag{4.23}$$

A similar derivation can be completed to find $\bar{E}[D^2(j^*)]$ for the second agent.

4.4.3 Difference Payoff Matrix Theory

We now prove that under certain conditions, difference evaluation functions provide a higher normalized expected payoff for the optimal action than the overall system evaluation does.

Theorem 1. *If :*

$$E[C^1(i^*)] < c_{avg} \tag{4.24}$$

$$E[C^2(j^*)] < c_{avg} \tag{4.25}$$

Then:

$$\bar{E}[D^1(i^*)] > \bar{E}[C^1(i^*)] \tag{4.26}$$

$$\bar{E}[D^2(j^*)] > \bar{E}[C^2(j^*)] \tag{4.27}$$

This means that if the expected payoff of taking the optimal action is less than the average of all possible payoffs, then the normalized expected payoff of the optimal action is higher when using the difference evaluation than when using the system

evaluation function. This leads to an improved basin of attraction for the optimal Nash equilibrium point, improving the probability of reaching this point.

Proof. We show that the conditions in Equations 4.24 and 4.25 leads to the normalized expected payoff for the optimal action being higher when using difference evaluations rather than the system evaluation function. We begin with our assumption for the first agent, given by Equation 4.24:

$$\begin{aligned} E[C^1(i^*)] &< c_{avg} \\ \Rightarrow E[C^1(i^*)] \cdot \left(\frac{c_{avg}}{c_{max}} - 1\right) &> \frac{c_{avg}^2}{c_{max}} - c_{avg} \end{aligned} \tag{4.28}$$

Note that $\left(\frac{c_{avg}}{c_{max}} - 1\right)$ is strictly negative from assumption 1, so the sign of the inequality changes. We thus have:

$$\begin{aligned} E[C^1(i^*)] \cdot \frac{c_{avg}}{c_{max}} - \frac{c_{avg}^2}{c_{max}} + c_{avg} &> E[C^1(i^*)] \\ \Rightarrow c_{max}\left(E[C^1(i^*)] \cdot \frac{c_{avg}}{c_{max}} - \frac{c_{avg}^2}{c_{max}} + c_{avg}\right) &> E[C^1(i^*)] \cdot c_{max} \\ \Rightarrow E[C^1(i^*)] \cdot c_{avg} - c_{avg}^2 + c_{avg} \cdot c_{max} &> E[C^1(i^*)] \cdot c_{max} \\ \Rightarrow n \cdot c_{avg} \cdot \big(E[C^1(i^*)] - c_{avg} + c_{max}\big) &> E[C^1(i^*)] \cdot n \cdot c_{max} \\ \frac{n \cdot c_{avg} \cdot (E[C^1(i^*)] - c_{avg} + c_{max})}{n^2 \cdot c_{max} \cdot c_{avg}} &> \frac{E[C^1(i^*)] \cdot n \cdot c_{max}}{n^2 \cdot c_{avg} \cdot c_{max}} \\ \Rightarrow \frac{E[C^1(i^*)] - c_{avg} + c_{max}}{n \cdot c_{max}} &> \frac{E[C^1(i^*)]}{n \cdot c_{avg}} \\ \Rightarrow \bar{E}[D^1(i^*)] &> \bar{E}[C^1(i^*)] \end{aligned}$$

An analogous derivation yields a similar result for the second population, so we have:

$$E[C^1(i^*)] < c_{avg} \Rightarrow \bar{E}[D^1(i^*)] > \bar{E}[C^1(i^*)]$$
$$E[C^2(j^*)] < c_{avg} \Rightarrow \bar{E}[D^2(j^*)] > \bar{E}[C^2(j^*)]$$

Thus, if the conditions from Equations 4.24 and 4.25 hold, then the normalized expected payoff of the optimal action is higher when using difference evaluations than it is when using the system evaluation function. □

4.4.4 Analysis of Theoretical Results

We have shown that the difference matrices yield a higher normalized expected payoff for the optimal action if the conditions from Equations 4.24 and 4.25 are met. This means that the difference evaluation either expands the basin of attraction around the optimal equilibrium point or increases the gradient leading into this point, improving system performance. For example, the difference evaluation function leads to a higher normalized expected payoff for the optimal action than the system evaluation does in the following game:

$$C_{difficult} = \begin{pmatrix} 45 & -250 & 45 \\ -250 & 50 & -250 \\ 45 & -250 & 45 \end{pmatrix} \tag{4.29}$$

However, the difference evaluation function does not lead to a higher normalized expected payoff for the optimal action than the system evaluation does in games such as:

$$C_{easy} = \begin{pmatrix} 45 & 49 & 45 \\ 49 & 50 & 49 \\ 45 & 49 & 45 \end{pmatrix} \tag{4.30}$$

In these games, the optimal Nash equilibrium is at $(2, 2)$. In the case of the game given by Equation 4.29, the optimal Nash equilibrium is difficult to reach, because if only one agent takes the optimal action, then the joint payoff is severely suboptimal. In the case of the game given by Equation 4.30, the optimal Nash equilibrium is very easy to reach, because the expected values of the optimal action for each agent are very high, regardless of the action choice of the collaborating agent. So, we see that for difficult games with deceptive optimal Nash equilibria, the difference evaluation function provides better feedback to learning agents than the system evaluation function, improving the probability that the optimal Nash equilibrium will be reached.

We now will examine how difference evaluations affect the payoff matrix in games where the conditions from Equations 4.24 and 4.25 hold, and we expect difference evaluations to improve system performance. The difference payoff matrix for the first agent for the game defined by Equation 4.29 is:

$$D^1_{difficult} = \begin{pmatrix} 295 & 0 & 295 \\ 96.7 & 396.7 & 96.7 \\ 295 & 0 & 295 \end{pmatrix} \tag{4.31}$$

We see that the difference payoff matrix allows for the agent to more easily reach the optimal Nash equilibrium at $(2, 2)$. The large penalties corresponding with one agent taking a suboptimal action have been relaxed, resulting in a less deceptive optimal Nash equilibrium. Further, the average payoffs for each action that the first agent can take are now all equivalent. The difference evaluation thus alters deceptive Nash equilibrium points by reducing penalties associated with one agent deviating from its best response.

If only one of the conditions from Equations 4.24 and 4.25 are met, then the probability of converging to the optimal Nash equilibrium depends on the extent to which the conditions are met or violated. In this case, the probability of convergence will vary based on the specific domain being analyzed.

So we have one of the following three cases, whether or not our assumptions in Equations 4.24 and 4.25 hold:

Case I: Equations 4.24 and 4.25 hold, and the difference evaluations yield a higher normalized expected payoff for the optimal action than the system evaluation function.

Case II: Neither Equation 4.24 or 4.25 holds, and the normalized expected payoff

of the optimal action when using difference evaluations is less than or equal to those when using the system evaluation function.

Case III: Only one of Equations 4.24 and 4.25 hold. The value of the normalized expected payoff of the optimal action when using difference evaluations will depend on how strongly the assumptions are held or violated.

In summary, the conditions in Equations 4.24 and 4.25 holding lead to a higher normalized expected payoff for the optimal action when using difference evaluations, improving the probability of reaching the optimal Nash equilibrium. If neither assumption holds, then the normalized expected payoff of the optimal action is greater when using the system evaluation function. If only one condition holds, then performance will depend on how strongly each condition is held or violated, and performance will vary by domain.

4.5 Empirical Results

We now compare the basins of attraction when using the system payoff matrix and the difference payoff matrices in the penalty game, defined as [91]:

$$C = \begin{pmatrix} 10 & 0 & p \\ 0 & 2 & 0 \\ p & 0 & 10 \end{pmatrix} \tag{4.32}$$

where p is a penalty term. The penalty game has three equilibrium points, at $(1,1)$, $(2,2)$, and $(3,3)$. The optimal Nash equilibria are located at $(1,1)$ and $(3,3)$. The value of the penalty p dictates how deceptive these equilibrium points are. If p is very low, then the expected payoffs of actions 1 and 3 are low, making it difficult to reach the optimal equilibrium points. If the value of p is high, then the optimal Equilibrium points are easy to reach. If $p < -8$, then the conditions from Equations 4.24 and 4.25 are met, and we expect difference evaluations to improve system performance. If $p > -8$, then the conditions are not met, and we do not expect difference evaluations to improve system performance.

Figures 4.1 and 4.2 show the basins of attraction for the system evaluation function and the difference evaluation functions when the conditions in Equations 4.24 and 4.25 hold and are violated, respectively. For a 3x3 game, each population is a vector containing three elements, where each element correlates to the probability of selecting the corresponding action. The x-axis of each plot is the initial value of the first element x_1 of the population vector, and the y-axis of each plot is the initial value of the second element x_2 of the population vector. Note that the third element x_3 is $1 - x_1 - x_2$.

As seen in Figure 4.1 when $p = -15$ and the conditions are met, the basin of attraction corresponding to the difference evaluation is much larger than when using the system evaluation function. Specifically, the difference evaluation provides a benefit when the second action (corresponding to the suboptimal Nash equilibrium) has a high probability of being taken. In these cases, the system evaluation function leads agents to converge to the suboptimal Nash equilibrium,

while the difference evaluation allows the agents to learn to find the optimal Nash equilibrium. The difference payoff matrix for the first agent when $p = -15$ is:

$$D_1|_{p=-15} = \begin{pmatrix} 11.67 & -0.67 & -13.33 \\ 1.67 & 1.33 & 1.67 \\ -13.33 & -0.67 & 11.67 \end{pmatrix} \tag{4.33}$$

The difference payoff matrix for the first agent when $p = 0$ is:

$$D_1|_{p=0} = \begin{pmatrix} 6.67 & -0.67 & -3.33 \\ -3.33 & 1.33 & -3.33 \\ -3.33 & -0.67 & 6.67 \end{pmatrix} \tag{4.34}$$

In cases where the optimal Nash equilibrium is deceptive (the conditions from Eq. 4.24 and 4.25 hold), the difference evaluation relaxes the penalty associated with the collaborating agent taking a suboptimal action, allowing for the optimal Nash equilibrium to be more easily reached. As seen in Equation 4.33, the suboptimal Nash equilibrium corresponding with the joint action $(2, 2)$ is no longer a Nash equilibrium, allowing for agents using difference evaluations to easily reach the optimal policy. When the conditions from Equations 4.24 and 4.25 do not hold, the difference payoff matrix for the first agent given in Equation 4.34 still contains the suboptimal Nash equilibrium.

As seen in Figure 4.2 when $p = 0$ and the conditions from Equations 4.24 and 4.25 are not met, the basin of attraction when using difference evaluations is

approximately equal to that using the system evaluation function. In this case, the suboptimal Nash equilibrium at $(2, 2)$ is found if an agent has a high probability of selecting the second action. The collaborating agent learns to also select the second action in order to maximize the payoff given the first agent's action selection.

These results are consistent with the theoretical results from Section 4.4.3, as well as the insight gleaned from examining the payoff matrices in Equations 4.33 and 4.34. When the conditions from Equations 4.24 and 4.25 hold, then the normalized expected payoff for the optimal action is higher when using difference evaluations rather than the system evaluation, and the basin of attraction corresponding to the optimal Nash equilibrium point is expanded. When the conditions do not hold, using the system evaluation function results in slightly better performance than when using difference evaluations. A particularly interesting result is in the case where the optimal Nash equilibrium point is extremely deceptive, the suboptimal Nash equilibrium is eliminated by the difference evaluation function, allowing for the optimal equilibrium point to be more easily reached.

4.6 Extension to Multiple Agents

We now extend the theoretical results to systems with three or more agents. This analysis is restricted to cooperative coevolutionary algorithms with n agents learning in a stateless domain, where each agent has a finite number of actions. As with the two agent case, this model assumes that the populations are infinite, and that the proportions of individuals in the populations are computed at each time step

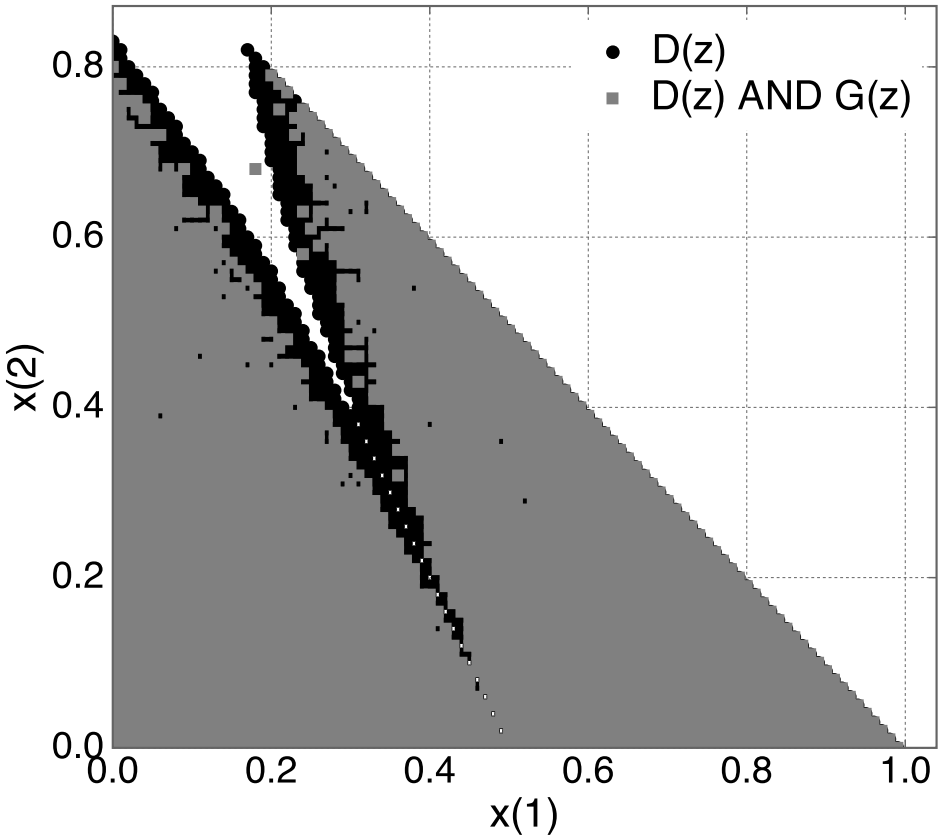

Figure 4.1: Case I (assumptions met): penalty game with $p = -15$. The basin of attraction corresponding to the difference payoff matrices is much larger than when using the system payoff matrix

during evolution. The number of actions that agent a may take is denoted l_a. Agent a's population at each generation is an element of $\Delta^{l_a} = \left\{ p \in [0,1]^{l_a} \,|\, \sum_{i=1}^{l_a} p_i = 1 \right\}$. The system payoff is defined by an n-dimensional array rather than a payoff matrix, as there are n (instead of two) agents. Each agent has a two-dimensional payoff matrix $C^a \in \mathbb{R}^{l_a \cdot \Pi_{i \neq a} l_i}$. The first dimension l_a corresponds to the action taken by agent i, and the second dimension $\Pi_{i \neq a} l_i$ corresponds to the joint action taken by

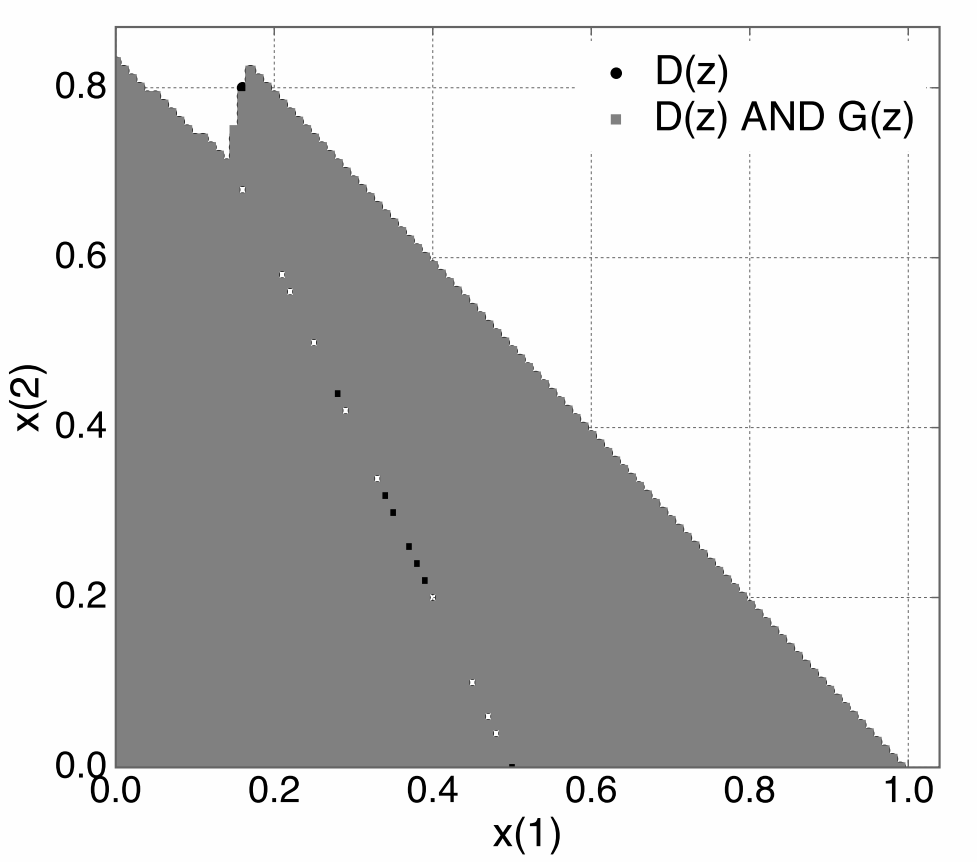

Figure 4.2: Case II (assumptions not met): penalty game with $p = 0$. The basins of attraction corresponding to the difference payoff matrices and system payoff matrix are approximately equal.

the rest of the agents in the system. The EGT model for CCEAs containing n agents is defined as:

$$u_i^a(t) = \sum_{j=1}^{\Pi_{k \neq i} l_k} c_{ij}^a y_j^a(t) \tag{4.35}$$

$$x_i^a(t+1) = \left(\frac{u_i^a(t)}{\sum_{k=1}^{l_a} x_k^a(t) u_k^a(t)} \right) x_i^a(t) \tag{4.36}$$

where:

- $u_i^a(t)$ is the fitness of agent a when taking action i at timestep t
- c_{ij}^a is the (i, j) element of agent a's payoff matrix
- $y_j^a(t)$ is the probability of the population (excluding agent a) takes joint action j
- $x_i^a(t)$ is the proportion of agent a's population which selects action i at timestep t

Equation 4.35 assigns fitness values to agent a's population members and Equation 4.36 updates the population of agent a. Now that we have defined the EGT model for multiple agents, we can analyze difference evaluation functions in such a system.

4.6.1 Difference Payoff Matrices and EGT Model

Consider the case where each agent has a system payoff matrix C^a, and we want D^a to be agent a's payoff matrix defined by the difference evaluation function. We assume the counterfactual term in the difference evaluation function is the average payoff for an agent across all actions, given the joint action of the rest of the agents in the system. Thus we have:

$$d_{ij}^a = c_{ij}^a - \frac{\sum_{k=1}^{l_a} c_{kj}^a}{l_a} + \alpha \tag{4.37}$$

The payoff matrix defined above is the difference payoff matrix for agent a. As in the two agent case, α is a constant to ensure that all payoff matrix elements are positive. The multiagent EGT model for agents using difference payoff matrices is:

$$u_i^a(t) = \sum_{j=1}^{\Pi_{k \neq i} l_k} d_{ij}^a y_j^a(t) \tag{4.38}$$

$$x_i^a(t+1) = \left(\frac{u_i^a(t)}{\sum_{k=1}^{l_a} x_k^a(t) u_k^a(t)} \right) x_i^a(t) \tag{4.39}$$

Note that this EGT model is identical to that defined in Equations 4.35 and 4.36, except the fitness of each agent is assigned using difference payoff matrices rather than the system payoff matrix.

4.6.2 Expected Payoffs Theory

In this section, we present the results of a theoretical analysis on the effectiveness of difference payoff matrices and global payoff matrices in multiagent systems of arbitrary size, using an evolutionary game-theoretic setting. For the analysis, we make the following assumptions:

Assumption I: All elements of the payoff matrix are positive

Assumption II: Each payoff matrix has at least two distinct elements (each element of the payoff matrix does not have the same value)

As in the two agent case, if assumption 1 is violated, we add a constant term to each element of the payoff matrix to ensure that the payoff matrix elements are positive. Assumption 2 is necessary for the proof, but cases where this assumption does not hold are of no interest from a theoretical perspective, as all joint actions taken by the system would yield an equal payoff.

Global Payoff Matrix The expected payoff for agent a taking action i is:

$$E\left[C^a(i)\right] = \frac{1}{\Pi_{k\neq a} l_k} \sum_{j=1}^{\Pi_{k\neq a} l_k} c_{ij}^a \tag{4.40}$$

where C^a is the payoff matrix for the first agent. The normalized expected payoff for agent a taking the optimal action $\bar{E}[C^a(i^{a*})]$ is the expected payoff of agent a's optimal action i^{a*} divided by the sum of expected payoffs for all possible actions:

$$\bar{E}[C^a(i^{a*})] = \frac{E[C^a(i^{a*})]}{\sum_{i=1}^{l_a} E[C^a(i)]} \tag{4.41}$$

$$= \frac{E[C^a(i^{a*})]}{\sum_{i=1}^{l_a} \left[\frac{1}{\Pi_{k\neq a} l_k} \sum_{j=1}^{\Pi_{k\neq a} l_k} c_{ij}^a\right]} \tag{4.42}$$

$$= \frac{E[C^a(i^{a*})]}{l_a c_{avg}} \tag{4.43}$$

where c_{avg} is the average of the system payoff matrix, defined by:

$$c_{avg} = \frac{1}{l_a} \sum_{i=1}^{l_a} \left[\frac{1}{\Pi_{k\neq a} l_k} \sum_{j=1}^{\Pi_{k\neq a} l_k} c_{ij}^a\right] \tag{4.44}$$

Note that c_{avg} is equivalent for all agents, as each agent's payoff matrix C^a is comprised of the same elements in different arrangements.

Difference Payoff Matrix The expected payoff for agent a when using the difference payoff matrix and taking action i is:

$$E[D^a(i)] = \frac{1}{\Pi_{k\neq a} l_k} \sum_{j=1}^{\Pi_{k\neq a} l_k} d^a_{ij} \tag{4.45}$$

$$= \frac{1}{\Pi_{k\neq a} l_k} \sum_{j=1}^{\Pi_{k\neq a} l_k} \left(c^a_{ij} - \frac{\sum_{k=1}^{l_a} c^a_{kj}}{l_a} + \alpha \right) \tag{4.46}$$

$$= E[C^a(i)] - c_{avg} + \alpha \tag{4.47}$$

The normalized expected payoff for agent a while taking the optimal action and using difference payoff matrices is defined as:

$$\bar{E}[D^a(i^{a*})] = \frac{E[D^a(i^{a*})]}{\sum_{i=1}^{l_a} E[D^a(i)]} \tag{4.48}$$

$$= \frac{E[C^a(i^{a*})] - c_{avg} + \alpha}{\sum_{i=1}^{l_a} (E[C^a(i)] - c_{avg} + \alpha)} \tag{4.49}$$

$$= \frac{E[C^a(i^{a*})] - c_{avg} + \alpha}{\sum_{i=1}^{l_a} (E[C^a(i)]) - l_a \cdot c_{avg} + l_a \cdot \alpha} \tag{4.50}$$

$$= \frac{E[C^a(i^{a*})] - c_{avg} + \alpha}{l_a \cdot \alpha} \tag{4.51}$$

Setting $\alpha = c_{max}$ ensures that the difference payoff matrix elements are all positive, so we have:

$$\bar{E}[D^a(i^{a*})] = \frac{E[C^a(i^{a*})] - c_{avg} + c_{max}}{l_a \cdot c_{max}} \tag{4.52}$$

where c_{max} is defined as:

$$c_{max} = max\left\{c_{ij}^a | i \in [1, l_a], j \in [1, \Pi_{k \neq a} l_k]\right\} \tag{4.53}$$

Note that the value of c_{max} is identical for each agent.

Difference Payoff Matrix Theory We now prove that under certain conditions, difference evaluation functions provide a higher normalized expected payoff for the optimal action than the overall system evaluation does.

Theorem 2. *If :*

$$E[C^a(i^{a*})] < c_{avg} \quad \forall a \tag{4.54}$$

Then:

$$\bar{E}[D^a(i^{a*})] > \bar{E}[C^a(i^{a*})] \quad \forall a \tag{4.55}$$

This means that if the expected payoff of each agent when taking the optimal action is less than the average of all possible payoffs, then the normalized expected

payoff of the optimal action is higher when using the difference evaluation than when using the system evaluation function, This leads to an improved basin of attraction for the optimal Nash equilibrium point, improving the probability of reaching this point.

Proof. We begin with the assumption from Equation 4.54 for agent a:

$$\begin{aligned} E[C^a(i^{a*})] &< c_{avg} \\ \Rightarrow E[C^a(i^{a*}] \cdot \left(\frac{c_{avg}}{c_{max}} - 1\right) &> \frac{c_{avg}^2}{c_{max}} - c_{avg} \end{aligned}$$

Note that $\left(\frac{c_{avg}}{c_{max}} - 1\right)$ is strictly negative from assumption 1, so the sign of the inequality changes. We thus have:

$$\begin{aligned} E[C^a(i^{a*})] \cdot \frac{c_{avg}}{c_{max}} - \frac{c_{avg}^2}{c_{max}} + c_{avg} &> E[C^a(i^{a*})] \\ \Rightarrow c_{max}\left(E[C^a(i^{a*})] \cdot \frac{c_{avg}}{c_{max}} - \frac{c_{avg}^2}{c_{max}} + c_{avg}\right) &> E[C^a(i^{a*})] \cdot c_{max} \\ \Rightarrow E[C^a(i^{a*})] \cdot c_{avg} - c_{avg}^2 + c_{avg} \cdot c_{max} &> E[C^a(i^{a*})] \cdot c_{max} \\ \Rightarrow l_a \cdot c_{avg} \cdot (E[C^a(i^{a*})] - c_{avg} + c_{max}) &> E[C^a(i^{a*})] \cdot l_a \cdot c_{max} \\ \Rightarrow \frac{l_a \cdot c_{avg} \cdot (E[C^a(i^{a*})] - c_{avg} + c_{max})}{l_a^2 \cdot c_{max} \cdot c_{avg}} &> \frac{E[C^a(i^{a*})] \cdot l_a \cdot c_{max}}{l_a^2 \cdot c_{avg} \cdot c_{max}} \\ \Rightarrow \frac{E[C^a(i^{a*})] - c_{avg} + c_{max}}{l_a \cdot c_{max}} &> \frac{E[C^a(i^{a*})]}{l_a \cdot c_{avg}} \\ \Rightarrow \bar{E}[D^a(i^{a*})] &> \bar{E}[C^a(i^{a*})] \end{aligned}$$

This derivation can be repeated for each agent, so we have:

$$E[C^a(i^{a*})] < c_{avg} \Rightarrow \bar{E}[D^a(i^{a*})] > \bar{E}[C^a(i^{a*})] \quad \forall a$$

Thus, if the condition from Equation 4.54 holds, then the normalized expected payoff of the optimal action is higher when using difference evaluations than it is when using the system evaluation function. □

We have demonstrated that the theoretical advantages of difference evaluations extend to multiagent systems with more than two agents. In cases where the condition from Equation 4.54 holds, the optimal Nash equilibrium is deceptive, because an agent taking the optimal action receives a low payoff unless each collaborating agent in the system also simultaneously take optimal actions. In these cases, the optimal Nash equilibrium is difficult to reach, and difference evaluations improve the basin of attraction around such equilibrium points, improving the probability of agents finding optimal policies.

4.7 Summary

In this chapter, we incorporated difference payoff matrices into the EGT model, and derived conditions under which using difference evaluations yields a higher normalized expected payoff for the optimal action than when using the system evaluation function. When these conditions are met, difference evaluations result in significant performance increases over using the system evaluation function.

When these conditions are not met, using the system evaluation function results in approximately equivalent performance as the difference evaluation function.

Conceptually, difference evaluations relax penalties associated with collaborating agents selecting suboptimal actions, as well as removing suboptimal Nash equilibrium points in some cases. Although difference evaluations alter the feedback that each agent receives, this alteration is not hand-tuned. Rather, the difference payoff matrices are simply the result of directly applying the difference evaluation function to the system payoff matrix.

The difference evaluation function has been used successfully in many CCEAs and multiagent reinforcement learning domains. Empirical results have shown that the difference evaluation function performs very well in solving the credit assignment problem, providing successful coordination in many cooperative multiagent systems. However, there has been no prescriptive theoretical work defining how the difference evaluation function affects the basin of attraction around the optimal Nash equilibrium.

The key contribution of this chapter is to provide the theoretical analysis which derives conditions under which the difference evaluation improves the expected payoff of optimal actions, allowing for better learned policies to be obtained. Difference evaluations have been successful in many different multiagent learning settings, but there has been no previous work defining when they should improve performance. This chapter provides a prescriptive framework detailing when difference evaluations are expected to improve performance in cooperative coevolutionary learning in multiagent systems. Recall from Chapter 1 that the second

key research question involving difference evaluations was to provide a prescriptive theoretical analysis, in order to determine when they are expected to benefit system performance. Through an evolutionary game theoretic analysis, this chapter has addressed this key research topic.

Chapter 5 – Approximating Difference Evaluation Functions with Local Knowledge

5.1 Motivation

Difference evaluation functions[1] 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 [5, 39]. These evaluation functions have two key theoretical advantages. First, they are aligned with the overall system objective [5]. 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 [5].

Difference evaluation functions have a simple formulation. They are the *difference* between the system objective function and the *counterfactual*, or the value of the system objective function independent of the agent being evaluated. This gives

[1]This chapter based on "Approximating Difference Evaluations with Local Knowledge," M. Colby, W. Curran, C. Rebhuhn, and K. Tumer, in Proceedings of the 13th International Conference of Autonomous Agents and Multiagent Systems, 2014

an approximation to 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 system evaluation function, or the value of the system evaluation function independent of a particular agent.

There are two key reasons counterfactual evaluation functions are difficult to compute. First, calculating this term 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 calculate 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 [95]. The counterfactual system evaluation function term is estimated using a function approximator that is trained on previous observations involving 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 function approximator. 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-specific knowledge or global knowledge of the system state.

In this chapter, we present a generic and model-free approach to approximating the system evaluation function in order to estimate 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. The only global information needed in this approach 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.

In empirical tests, our results show that using this local approximation results in faster learning than using global evaluation functions for feedback, while attaining up to 98% of the performance when using difference evaluation functions calculated with global knowledge of the system. The key contribution of this work is to demonstrate the effectiveness of locally approximating difference evaluations to provide agent-specific feedback which improves coordination in multiagent systems.

Section 5.2 describes the approximation technique used in each domain. Section 5.3 presents the two different domains in which our reward approximation method

is applied. Section 5.4 shows the performance of our developed approximations in each domain, and in Section 5.5 we discuss the impact of these results.

5.2 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. 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 coordinate in order to maximize some system level objective function $G(\vec{s}, \vec{a})$, and that the true value of $G(\vec{s}, \vec{a})$ is available to each agent. Recall that the difference evaluation is defined as:

$$D_i(\vec{s}, \vec{a}) = G(\vec{s}, \vec{a}) - G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i}) \tag{5.1}$$

As we assume that $G(\vec{s}, \vec{a})$ is available to each agent, approximating the difference evaluation function only requires approximating the $G(\vec{s}_{-i} + \vec{c}_{s,i}, \vec{a}_{-i} + \vec{c}_{a,i})$ term. Given the particular domain, a suitable function approximator is chosen. For a stateless domain with discrete actions, this approximator may simply be a tabular function approximator. For a stateful domain with continuous actions, this

Algorithm 6: 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.

```
Initialize N agents
foreach Agent do
    Initialize private function approximator Ĝ_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
    Broadcast G(s⃗, a⃗) to each agent
    foreach Agent do
        Update Ĝ_i(s_i, a_i) using s_i, a_i, and G(s⃗, a⃗)
        D̂_i(s⃗, a⃗) = G(s⃗, a⃗) − Ĝ_i(c_{s,i}, c_{a,i})
        Update policy/value table based on D̂_i(s⃗, a⃗)
    end
end
```

approximator may be a neural network. Note that these are not the only possible options for these function approximators, but are potential choices which match the domains they will be used in.

The difference evaluation approximation algorithm is presented in Algorithm 6. At each time step, each agent takes an action, and $G(\vec{s}, \vec{a})$ is computed and broadcast to each agent. Each agent i maintains a private approximation for $G(\vec{s}, \vec{a})$, denoted as $\hat{G}_i(s_i, a_i)$. Note that the only information available to the agent includes the agent's state, action choice, and the broadcast value of $G(\vec{s}, \vec{a})$. Each agent's approximation of $G(\vec{s}, \vec{a})$ is therefore a mapping from that agent's state

and action to the value of the system objective function. It is of note that this approximation is initially very noisy, because the state and action information used to approximate $G(\vec{s}, \vec{a})$ 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 $G(\vec{s}, \vec{a})$ 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 $\hat{G}_i(s_i, a_i)$, the difference evaluation function can be estimated as:

$$\hat{D}_i(\vec{s}, \vec{a}) = G(\vec{s}, \vec{a}) - \hat{G}_i(c_{s,i}, c_{a,i}) \tag{5.2}$$

where $\hat{D}_i(\vec{s}, \vec{a})$ 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 at the beginning of each episode for evaluation. 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.*

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 contin-

uous 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.

5.2.1 Stateless Discrete Action Domains

Algorithm 7: Stateless Discrete-Action Multiagent Reinforcement Learning using $D_i(\vec{s}, \vec{a})$ Approximation

```
Initialize N agents
foreach Agent do
    Initialize private value table V_i(a)
    Initialize private function approximator Ĝ_i(a)
end
foreach Learning Step do
    foreach Agent do
        select action a_i using agent's policy
    end
    Broadcast G(a⃗) to each agent
    foreach Agent do
        Ĝ_i(a_i) ← α_1 · Ĝ_i(a_i) + (1 − α_1) · G(a⃗)
        D̂_i(a_i) = G(a⃗) − Ĝ_i(a_0)
        V_i(a_i) ← α_2 · V_i(a_i) + (1 − α_2) · D̂_i(a_i)
    end
end
```

We now demonstrate how Algorithm 6 may be implemented for a multiagent reinforcement learning algorithm in a stateless discrete action domain. The implementation of Algorithm 6 in a multiagent reinforcement learning problem with a

stateless discrete action domain is given in Algorithm 7. 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 $G(\vec{a})$, which is simply a table 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 $G(\vec{a})$ is then calculated, and this value is broadcast to each agent in the system. Each agent then updates its approximation $\hat{G}_i(a_i)$ according to:

$$\hat{G}_i(a_i) \leftarrow (1 - \alpha_1) \cdot \hat{G}_i(a_i) + \alpha_1 \cdot G(a) \tag{5.3}$$

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

$$\hat{D}_i(a_i) = G(\vec{a}) - \hat{G}_i(c_{a,i}) \tag{5.4}$$

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

$$Q_i(a_i) \leftarrow (1 - \alpha_2) \cdot Q_i(a_i) + \alpha_2 \cdot \hat{D}_i(a_i) \tag{5.5}$$

5.2.2 Continuous State and Action Domains

We now demonstrate how Algorithm 6 may be implemented in a cooperative coevolutionary algorithm in a continuous state and action domain. Note that such problems may also be solved using multiagent reinforcement learning, and we are simply demonstrating an approach incorporating evolutionary algorithms in order to demonstrate that difference evaluation approximations may be used in both reinforcement learning and evolutionary algorithm settings.

We assume each agent has a neural network policy which maps the agent's state to an action. Further, each agent has a two-layer feedforward neural network approximation of $G(\vec{s}, \vec{a})$. 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 $G(\vec{s}, \vec{a})$ 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 $(s, a, G(\vec{s}, \vec{a}))$ tuple and backpropogation, where the error in the approximation is $G(\vec{s}, \vec{a}) - \hat{G}_i(s_i, a_i)$.

The difference evaluation is then approximated using Equation 5.2, and the estimated value of $\hat{D}_i(s_i, a_i)$ 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 6 using a cooperative coevolutionary algorithm is given in Algorithm 8.

5.3 Experimental Domains

In this section, we introduce the El Farol bar problem, and provide a detailed description of the system dynamics and evaluation functions used in this domain. Our approximation approach is tested in two domains. 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, and was presented in Section 3.3. These domains were chosen to highlight that our approximation approach can be applied in a wide range of different multiagent problems.

5.3.1 Bar Problem

The El Farol Bar Problem [9] 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 which 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:

$$L_i = e^{\frac{-x_i(a_i)}{c}} \tag{5.6}$$

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:

$$G(\vec{a}) = \sum_{k=0}^{K} x_k(\vec{a}) e^{\frac{-x_k(\vec{a})}{c}} \tag{5.7}$$

where k is the index of the night, and $x_k(\vec{a})$ is the number of agents who attended on the kth night. From the reward, we know that if there are enough agents to be equally spread out across the bars, $n \leqslant c \cdot k$, this becomes a scheduling problem. This problem becomes a congestion problem when there are more than twice as many agents as the capacity for each night allows, $n > 2c \cdot k$. In this case the optimal response is for the majority of agents to attend one night, thus making agents attending that night receive a very low reward, and the rest of the agents equally distributing over the rest of the nights such that the number of agents for each other night becomes c, receiving the optimal reward for those nights.

5.4 Experimental Results

The following sections detail the experimental results in both 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.

5.4.1 Bar Problem Domain Results

The bar problem was initialized as follows. There are 1000 agents and 10 nights, where each night had an optimal capacity of 10. The learning rate α_1 for the approximation of the system evaluation function is set to 0.1. The learning rate α_2 for the value table is set to 0.1. Each experimental run lasted for $500,000$ timesteps, and there were 100 statistical runs conducted. The experimental results are shown in Figure 5.1, and the reported error bars are error in the mean σ/N^2.

As seen in Figure 5.1, approximating the difference evaluation function results in almost 10 times better performance than using the system evaluation function $G(\vec{a})$. 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. Consider the case where the system was a "black box," in which agent actions were inputs and the system evaluation function $G(\vec{a})$ was an output. In this case, no domain knowledge is present, and it is impossible to analytically determine $G(\vec{a})$. 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.

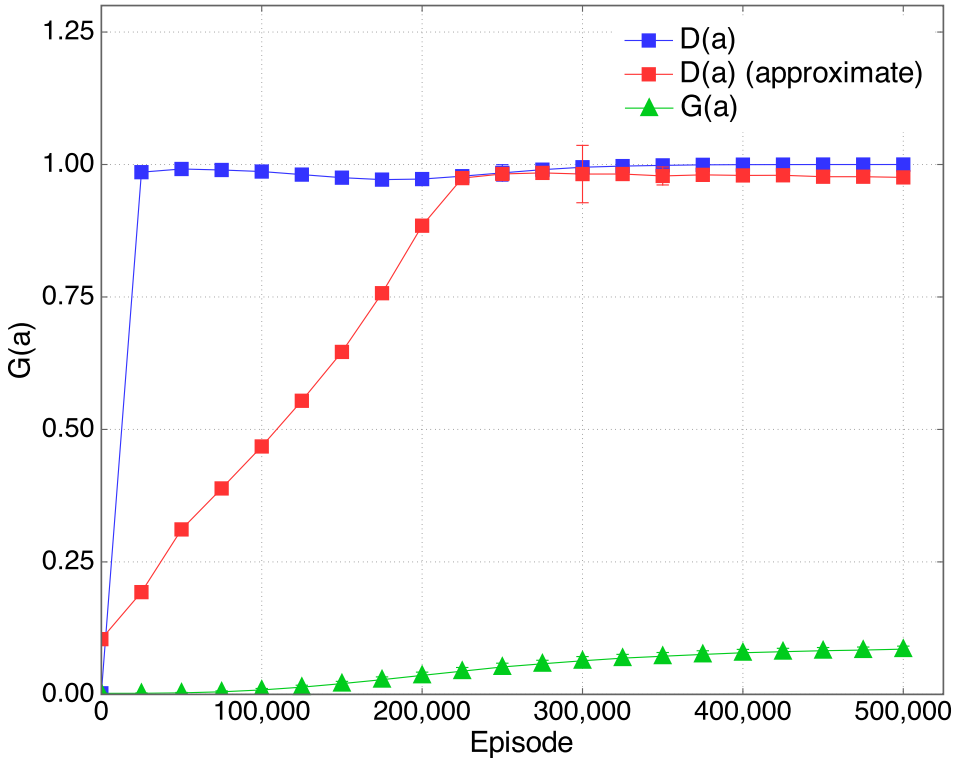

Figure 5.1: Bar problem results. When approximating $D_i(a)$, the learning rate is slower than when using $D_i(a)$, but the converged performance is nearly identical. $D_i(a)$ performs almost 10 times better than the overall system evaluation function $G(\vec{a})$.

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.

5.4.2 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 coevolutionary algorithm was run for 3000 generations, and 150 statistical runs were conducted for statistical significance. The experimental results are shown in Figure 5.2. As in the bar problem, reported error bars represent the error in the mean of the statistical data.

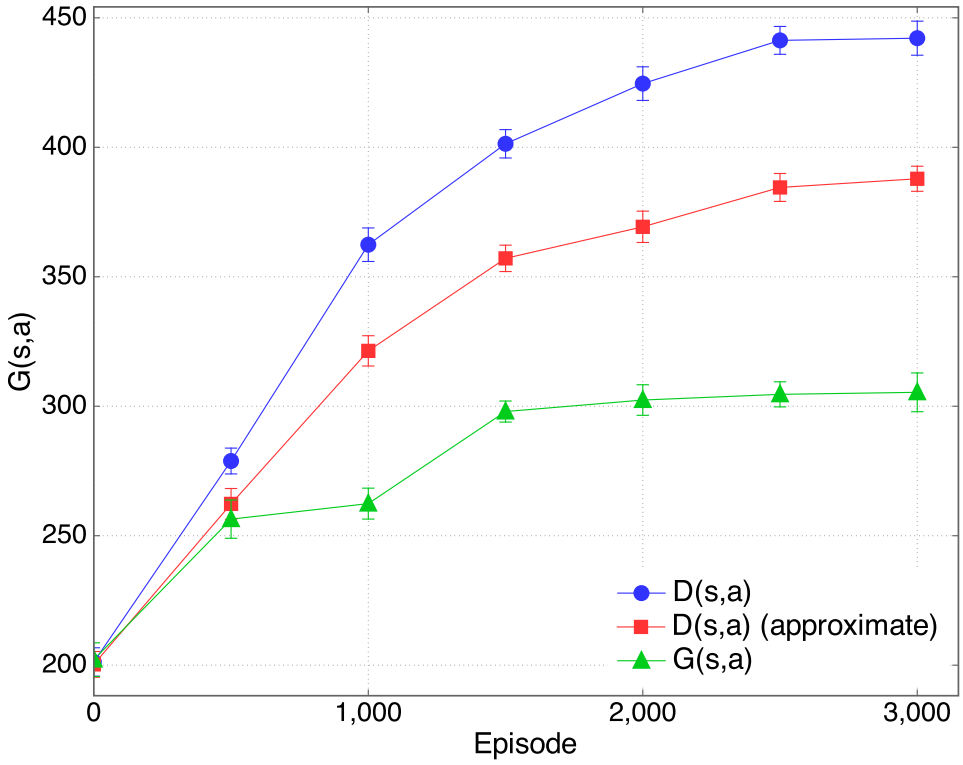

Figure 5.2: Rover domain results (10 agents). Approximating $D_i(s,a)$ results in 88% of the performance attained when analytically computing $D_i(s,a)$. Approximating the difference evaluation function results in significant performance gains when compared to using the system evaluation function $G(\vec{s},\vec{a})$.

As seen in Figure 5.2, approximating $D_i(\vec{s},\vec{a})$ results in approximately a 23% increase in converged performance compared to when using the system evaluation function $G(\vec{s},\vec{a})$. When approximating $D_i(\vec{s},\vec{a})$, the converged performance was 88% of the converged performance when analytically computing $D_i(\vec{s},\vec{a})$. 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 $D_i(\vec{s},\vec{a})$ requires only an agent's local

state and action, and the value of the system evaluation function $G(\vec{s}, \vec{a})$. In order to analytically compute $D_i(\vec{s}, \vec{a})$, global information about the state and action of each agent is required, as well as the function $G(\vec{s}, \vec{a})$. So, although estimating $D_i(\vec{s}, \vec{a})$ results in slightly worse performance than analytically computing $D_i(\vec{s}, \vec{a})$, 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 $D(\vec{s}, \vec{a})$ 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 complicated than in the bar problem, and explains why the approximation resulted in decreased performance of $\hat{D}_i(s, a)$ when compared to $D(\vec{s}, \vec{a})$.

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

5.4.3 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 5.4.2. The experimental results are shown in Figure 5.3. The reported error bars represent the error in the mean of the statistical data.

As seen in Figure 5.3, approximating the difference evaluation function leads to 79% of the performance when using the analytically computed difference evaluation, outperforms the system evaluation function by 49%. Although $\hat{D}_i(s,a)$ performed worse compared to $D_i(\vec{s},\vec{a})$ in this larger domain (79% vs. 88%), it outperformed $G(\vec{s},\vec{a})$ 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.

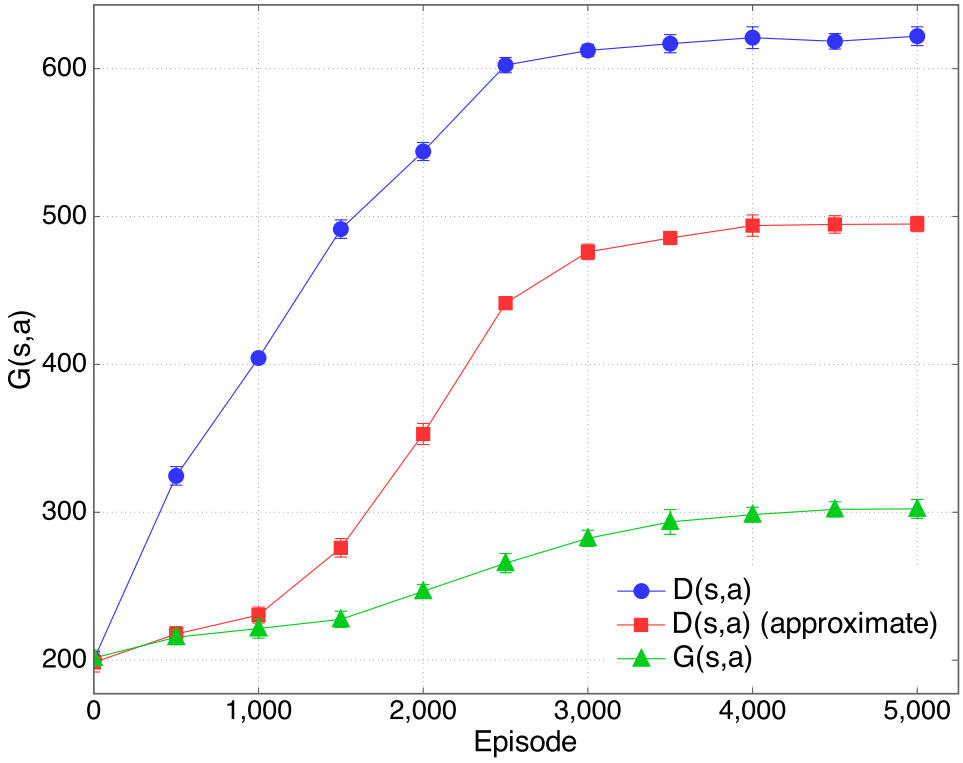

Figure 5.3: Rover domain results (100 agents). Approximating $D_i(\vec{s}, \vec{a})$ results in 79% of the performance attained when analytically computing $D_i(\vec{s}, \vec{a})$. Approximating the difference evaluation function results in significant performance gains when compared to using the system evaluation function $G(\vec{s}, \vec{a})$.

It is of note that in this larger domain, learning occurs more slowly (as compared to $D_i(\vec{s}, \vec{a})$) than in the 10 agent domain. This can be attributed to the fact that the approximation of $G\vec{s}, \vec{a})$ 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 $D_i(\vec{s}, \vec{a})$, they still significantly outperform agents using $G(\vec{s}, \vec{a})$. 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.

5.4.4 Theoretical Implications of $D(z)$ Approximation

When approximating difference evaluations, each agent maintains an approximation of $G(\vec{s}, \vec{a})$, and uses this approximation to estimate the counterfactual. The difference evaluation is fully factored, meaning that an agent acting to increase the value of $D_i(\vec{s}, \vec{a})$ will always also increase the value of $G(\vec{s}, \vec{a})$. However, when approximating the difference evaluation, the guarantee of factoredness is lost. For the approximation of difference evaluations to be fully factored, we would have to demonstrate that the sign of $\hat{G}(s, a)$'s gradient is always equal to the sign of $G(\vec{s}, \vec{a})$'s gradient, which is not always possible. It is important to 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 of the system evaluation function, we performed random sampling in the rover domain to compare $G(\vec{s}, \vec{a})$ 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 $G(\vec{s}, \vec{a})$ 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 $G(\vec{s}, \vec{a})$ for this particular state-action tuple. This process was repeated for one million randomly generated states.

We found that $\hat{G}_i(s_i, a_i)$ was factored with $G(\vec{s}, \vec{a})$ 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 such good learning performance. Second, when the approximation of $G(\vec{s}, \vec{a})$ 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 $\hat{D}_i(s, a)$.

5.5 Summary

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 chapter, 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 $G(\vec{s}, \vec{a})$ can be broadcast to each agent, which in most cases is a reasonable assumption, and in fact is an assumption in all multiagent learning domains. 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 provided very interesting insight. 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 extremely 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 has not been fully investigated.

Recall from Chapter 1 that the third key research question about difference evaluations was how they may be implemented in systems where global state and action knowledge is unavailable to agents. In this chapter, we addressed this research question by introducing a novel approach for approximating difference evaluations with only local state and action knowledge. In any multiagent learning domain, it is typically assumed that the system performance $G(\vec{s}, \vec{a})$ is broadcast to each agent; further, it is assumed that each agent can determine its local state s_i and action a_i. Using only this knowledge, which is available in any multiagent learning system, we demonstrate that agents can successfully approximate difference evaluations, significantly improving system performance over using the system evaluation function for agent feedback.

Algorithm 8: Continuous State and Action CCEA using $D_i(s_i, a_i)$ Approximation

```
Initialize N populations of k neural networks
foreach Population i do
    foreach Individual j do
        Initialize function approximator Ĝ_{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 → 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_j
            each agent j in team T_i chooses action a_j
            G(s⃗, a⃗) is broadcast to each agent
            update Ĝ_{i,j}(s_i, a_i) based on s_i, a_i, G(s⃗, a⃗)
            D̂_{i,j}(s_j, a_j) = G(s⃗, a⃗) − Ĝ_{i,j}(c_{s,i}, c_{s,a})
            each agent j increments fitness by D̂_{i,j}(s_j, a_j)
        end
    end
    foreach Population do
        select k networks using ε-greedy
    end
end
```

Chapter 6 – Discussion

The automation of tasks through agent-based control is becoming more critical in real-world applications, as autonomous systems have significant advantages. Examples of the need for autonomy are numerous. Self-driving cars allow humans to work on more important tasks or relax as they are driven around, improving efficiency. Household robots to assist the elderly allow for autonomous agents to complete tasks that are difficult for the humans they serve, improving quality of life. Autonomous unmanned aerial systems can drastically improve coverage in search and rescue operations, increasing the likelihood that lost people can be found before a tragedy occurs. Creating autonomous agents to perform tasks improves system performance, and allows humans to spend time on different tasks.

Often, many autonomous agents are needed to achieve a complex task. For example, although a single autonomous Unmanned Aerial System (UAS) may be fitted with cameras and infrared sensors to assist in the search and rescue of a lost hiker in a national park, multiple autonomous UASs greatly improve the coverage of the environment, improving the odds of finding the lost hiker. Although more agents allow a task to be completed more efficiently, it is extremely difficult to develop control policies such that each agent acting autonomously will coordinate with other agents in the system to improve overall system performance. Multiagent learning techniques can develop such control policies, improving the coordination

and control of autonomous multiagent systems.

Cooperative coevolutionary algorithms involve training a set of autonomous agents to coordinate in order to achieve a system level objective. Developing good control policies for multiagent systems is a critical area of research, and is applicable to many real-world scenarios including air traffic control and distributed sensor network control. One of the key difficulties in multiagent learning systems is addressing the credit assignment problem. As multiple agents are interacting with each other and the environment, it is difficult to determine the effectiveness of any particular agent in the context of overall system performance. In order to learn good policies in a multiagent setting, the credit assignment problem must be addressed.

One solution to the credit assignment problem is the difference evaluation function, which approximates an individual agent's impact on the system evaluation function. The difference evaluation is aligned with the system evaluation function, and it has a favorable signal-to-noise ratio. These properties result in difference evaluation functions providing informative and beneficial agent-specific feedback during learning, resulting in improved learned performance. The performance benefits of difference evaluations have been demonstrated empirically in many domains, including congestion games, multiple robot coordination, air traffic control, and distributed sensor network control.

Although difference evaluation functions have nice theoretical properties and extensive empirical evidence supporting their usefulness, there are three research topics which have not been adequately addressed. First, although difference eval-

uations have provided good learned performance, it is unclear how extending difference evaluations and combining them with other coordination mechanisms will improve overall system performance. Second, there has been no prescriptive theoretical analysis of difference evaluations, providing conditions which describe scenarios where difference evaluations are expected to improve system performance over traditional coordination mechanisms. Third, there is not a clear path to implementation in scenarios where difference evaluations can not be directly computed, and agents only have access to local knowledge. This dissertation addressed each of these research questions, and demonstrates the real-world usefulness of difference evaluation functions.

The contributions of this dissertation are to:

1. Demonstrate difference evaluations are compatible with other coordination mechanisms, and combining difference evaluations with hall of fame approaches can improve system performance by up to 48%.

2. Provide a prescriptive theoretical result which derives conditions under which difference evaluations are beneficial for overall system performance.

3. Derive a technique for approximating difference evaluations with local information.

The first contribution demonstrates that difference evaluations are compatible with other types of coordination mechanisms, and pairing difference evaluations with these mechanisms can significantly improve system performance. We also

demonstrate that these combinations are extremely efficient with respect to computational resources, allowing for their implementation without a need for more sophisticated computational power.

The second contribution provides the first prescriptive theoretical framework for difference evaluations, deriving conditions under which difference evaluations are expected to improve system performance. The derived conditions demonstrate that difference evaluations are beneficial in systems with deceptive optimal policies. Thus, in systems where optimal policies are simple to find, most coordination mechanisms will provide adequate coordination of control of multiagent systems; however, in more difficult systems, difference evaluations allow for optimal policies to be found more easily. These theoretical conditions allow for a form of system identification to aid the system designer while determining which coordination mechanisms should be used to train a multiagent system.

The third contribution derives a technique for approximating difference evaluations in systems where agents only have access to local state and action information, as well as a broadcast value of the system evaluation function. This information is almost always available in multiagent learning settings, meaning that difference evaluation functions can be implemented in systems even when there isn't sufficient information to directly compute them. Thus, the advantages of difference evaluation functions may be attained even in systems where they can't be analytically computed.

By demonstrating that difference evaluations are compatible with other coordination mechanisms, we found that these pairings often produce excellent system

performance. The evolutionary game theoretic analysis provided a *prescriptive theoretical framework* which derived conditions under which difference evaluation functions are beneficial. The derivation of a novel technique to approximate difference evaluations provides a clear *path to implementation* of difference evaluations, allowing for their use in any generic multiagent system where the value of the system evaluation function is broadcast. By providing improved performance, a theoretical framework, and a clear path to implementation, we have demonstrated that difference evaluation functions are useful tools to be considered for real-world applications.

There are many avenues of potential future work from this research. An analysis of the compatibility of difference evaluations with other coordination mechanisms not considered in this dissertation could provide more insight on the general compatibility of difference evaluations and other techniques, as well as potentially provide even larger performance gains. The evolutionary game theoretic analysis could be extended to account for a combination of difference evaluation functions and other coordination mechanisms, rather than difference evaluation functions alone. The approximation technique could be extended to include more than information directly local to an agent. For example, in a large multiagent system, agents may often be able to communicate with agents which are spatially close. This information from other agents could be implemented in an approximation of difference evaluations, improving the accuracy of the approximation and subsequently improving the performance of learned policies.

Future work also includes new applications, specifically autonomous multiagent

control of advanced energy systems. New hybrid power plants which are extremely efficient are being developed, and traditional control techniques such as model predictive control are not capable of controlling these plants. We plan on applying multiagent learning techniques and difference evaluations to attempt to develop distributed controllers for these plants. Future work also includes more detailed investigations of phenomena we see in the various experiments we conducted on difference evaluations. For example, when approximating difference evaluations, there appeared to be a strong coupling between the degree of factoredness of the approximation and the performance level when using the approximation. If this coupling could be quantified, we could determine how accurate we need our approximation of difference evaluations to be in order to produce a specific level of system performance.

Bibliography

[1] J. Abounadi, D.P. 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, C. Holmes Parker, and K. Tumer. Evolving large scale uav communication systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Philadelphia, PA, July 2012.

[3] A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, pages 1–12, Seattle, WA, June 2004.

[4] A. Agogino and K. Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, pages 980–987, Washington, DC, USA, 2004. IEEE Computer Society.

[5] A. Agogino and K. Tumer. Regulating air traffic flow with coupled agents. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, May 2008.

[6] A. K. 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.

[7] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

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

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

[10] M. Babes, E.M. de Cote, and M.L. Littman. Social reward shaping in the prisoner's dilemma. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '08, pages 1389–1392, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[11] T. Bäck. Parallel optimization of evolutionary algorithms. In *In Parallel Problem Solving From Nature*, pages 418–427. Spring-Verlag, 1994.

[12] T. Back and H.P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1(1):1–23, March 1993.

[13] T. Balch and R.C. Arkin. Behavior-based formation control for multirobot teams. *Robotics and Automation, IEEE Transactions on*, 14(6):926–939, Dec 1998.

[14] K. Binmore. *Fun and Games, A text on game theory.* D.C. Heath and Company, 1992.

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

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

[17] M. Broom and C. Cannings. *Evolutionary Game Theory.* John Wiley And Sons, Ltd, 2001.

[18] A. Bucci and J.B. Pollack. On identifying global optima in cooperative coevolution. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 539–544, New York, NY, USA, 2005. ACM.

[19] A. Bucci and J.B. Pollack. Thoughts on Solution Concepts. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 434–439, New York, NY, USA, 2007. ACM.

[20] D. Büche and R. Dornberger. New evolutionary algorithm for multi-objective optimization and its application to engineering design problems. In *In Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization*, 2001.

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

[22] D. Büche, P. Stoll, R. Dornberger, and P. Koumoutsakos. Multi-objective evolutionary algorithm for the optimization of noisy combustion processes. *IEEE TRansaction on Systems, Man, and cybernetic*, 32:2002, 2002.

[23] M.D. Bugajska and A.C. Schultz. Coevolution of form and function in the design of micro air vehicles. In *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*, pages 154 – 163, 2002.

[24] T. Brgers and R. Sarin. Learning through reinforcement and replicator dynamics. *Journal of Economic Theory*, 77(1):1 – 14, 1997.

[25] G.P. Cachon and S. Netessine. Game theory in supply chain analysis. In D. Simchi-Levi, S.D. Wu, and Z.J. Shen, editors, *Handbook of Quantitative Supply Chain Analysis*, volume 74 of *International Series in Operations Research and Management Science*, pages 13–65. Springer US, 2004.

[26] Z. Cai and Z. Peng. Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems. *Journal of Intelligent and Robotic Systems*, 33(1):61–71, 2002.

[27] A. Canova, G. Gruosso, and M. Repetto. Magnetic design optimization and objective function approximation. *Magnetics, IEEE Transactions on*, 39(5):2154–2162, Sept 2003.

[28] L. Cardamone, D. Loiacono, and P.L. Lanzi. Applying cooperative coevolution to compete in the 2009 torcs endurance world championship. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, july 2010.

[29] D.S. Chen and R.C. Jain. A robust backpropagation learning algorithm for function approximation. *Neural Networks, IEEE Transactions on*, 5(3):467–479, May 1994.

[30] H. Chen and X. Wang. Cooperative coevolutionary algorithm for unit commitment. *Power Systems, IEEE Transactions on*, 17(1):128–133, Feb 2002.

[31] A. Coelho and D. Weingaertner. Evolving coordination strategies in simulated robot soccer. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 147–148, New York, NY, USA, 2001. ACM.

[32] M. Colby. Optimizing ballast design of wave energy converters using evolutionary algorithms. Master's thesis, Oregon State University, Corvallis, Oregon, 2012.

[33] M. Colby. Theory and applications of difference evaluation functions. In *In Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multiagent Systems, Doctoral Consortium (AAMAS-DC 13)*. Saint Paul, Minnesota, 2013.

[34] M. Colby, W. Curran, C. Rebhuhn, and K. Tumer. Approximating difference evaluations with local knowledge (extended abstract). In *In Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 14)*. Paris, France, 2014.

[35] M. Colby, M. Knudson, and K. Tumer. Multiagent flight control in dynamic environments with cooperative coevolutionary algorithms. In *In Association for the Advancement of Artificial Intelligence, Modeling in Human Machine Systems: Challenges for Formal Verification 2014*. Palo Alto, CA, 2014.

[36] M. Colby, N. Nasroullahi, and K. Tumer. Optimizing ballast design of wave energy converters using evolutionary algorithms. In *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 11)*. Dublin, Ireland, 2011.

[37] M. Colby, C. Holmes Parker, and K. Tumer. Coordination and control for large distributed sensor networks. In *In Future of Instrumentation International Workshop (FIIW-2012)*. Gatlinburg, TN, 2012.

[38] M. Colby and K. Tumer. Performance and fiscal analysis of distributed sensor networks in a power plant. In *In AAMAS-2012 Workshop on Agent Technologies for Energy Systems (ATES 12)*. Valencia, Spain, 2012.

[39] M. Colby and K. Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *AAMAS '12: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 425–432,

Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[40] M. Colby and K. Tumer. Multiagent reinforcement learning in a distributed sensor network with indirect feedback. In *The International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[41] M. Colby and K. Tumer. distributed sensor network control with evolutionary algorithms. In *In Future of Instrumentation International Conference (FIIC 2013)*. Orlando, FL, 2013.

[42] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1327–1332, 2002.

[43] F. Dan Foresee and M.T. Hagan. Gauss-newton approximation to bayesian learning. In *Neural Networks,1997., International Conference on*, volume 3, pages 1930–1935 vol.3, Jun 1997.

[44] E. de Jong, K.O. Stanley, and R.P. Wiegand. Introductory tutorial on coevolution. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, GECCO '07, pages 3133–3157, New York, NY, USA, 2007. ACM.

[45] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 225–232, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[46] S. Devlin, D. Kudenko, and M. Grzes. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278, 2011.

[47] S. Devlin and L. Ylienimi. Potential-based difference rewards for multiagent reinforcement learning. In *In Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*. 2014.

[48] M.B. Dias, Robert Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006.

[49] S. Ficici. Monotonic solution concepts in coevolution. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 499–506, New York, NY, USA, 2005. ACM.

[50] S.G. Ficici and J.B. Pollack. A game-theoretic approach to the simple coevolutionary algorithm. In Marc Schoenauer, Kalyanmoy Deb, Gnther Rudolph, Xin Yao, Evelyne Lutton, JuanJulian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 467–476. Springer Berlin Heidelberg, 2000.

[51] S.G. Ficici and J.B. Pollack. Pareto optimality in coevolutionary learning. In J. Kelemen and P. Sosk, editors, *Advances in Artificial Life*, volume 2159 of *Lecture Notes in Computer Science*, pages 316–325. Springer Berlin Heidelberg, 2001.

[52] D.B. Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3 –14, Jan 1994.

[53] D. Foster and P. Young. Stochastic evolutionary game dynamics. *Theoretical Population Biology*, 38(2):219 – 232, 1990.

[54] Y. Freund and R.E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, COLT '96, pages 325–332, New York, NY, USA, 1996. ACM.

[55] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez. Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks). *Neural Netw.*, 15(10):1259–1278, December 2002.

[56] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez. Covnet: a cooperative coevolutionary model for evolving artificial neural networks. *Neural Networks, IEEE Transactions on*, 14(3):575–596, May 2003.

[57] 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.

[58] B.P. Gerkey and M.J. Mataric. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768, Oct 2002.

[59] J.K. Goeree and C.A. Holt. Ten little treasures of game theory and ten intuitive contradictions. *The American Economic Review*, 91(5):pp. 1402–1422, 2001.

[60] C.K. Goh and K. Chen Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 13(1):103–127, Feb 2009.

[61] M. Grze and D. Kudenko. Multigrid reinforcement learning with reward shaping. In V. Krkov, R. Neruda, and J. Koutnk, editors, *Artificial Neural Networks - ICANN 2008*, volume 5163 of *Lecture Notes in Computer Science*, pages 357–366. Springer Berlin Heidelberg, 2008.

[62] J.C. Harsanyi and R. Selten. *A General Theory of Equilibrium Selection in Games*, volume 1 of *MIT Press Books*. The MIT Press, January 1988.

[63] T.D. Haynes, D.A. Schoenefeld, and R.L. Wainwright. Type inheritance in strongly typed genetic programming. In *Advances in Genetic Programming 2, chapter 18*, pages 359–376. MIT Press, 1996.

[64] P. J. Hoen and E. D. De Jong. Evolutionary multi-agent systems. In *In Proceedings of the 8th International Conference on Parallel Problem Solving from Nature PPSN-04*, pages 872–881, 2004.

[65] C.A. Iglesias, M. Garijo, J.C. Gonzlez, and J.R. Velasco. Analysis and design of multiagent systems using mas-commonkads. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV Agent Theories, Architectures, and Languages*, volume 1365 of *Lecture Notes in Computer Science*, pages 313–327. Springer Berlin Heidelberg, 1998.

[66] A.W. Iorio and X. Li. A cooperative coevolutionary multiobjective algorithm using non-dominated sorting. In K. Deb, editor, *Genetic and Evolutionary Computation GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 537–548. Springer Berlin Heidelberg, 2004.

[67] Y. Isukapalli. An Analytically Tractable Approximation for the Gaussian Q-Function. In *IEEE Communications Letters*, volume 12, pages 669 – 671.

[68] L. Jiao, J. Liu, and W. Zhong. An organizational coevolutionary algorithm for classification. *Evolutionary Computation, IEEE Transactions on*, 10(1):67–80, Feb 2006.

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

[70] Y. Jin and B. Sendhoff. Fitness approximation in evolutionary computation - a survey. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 1105–1112, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[71] M. Kearns, M. L. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 253–260, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[72] Y.K. Kim, J.Y. Kim, and Y. Kim. A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines. *Applied Intelligence*, 13(3):247–258, 2000.

[73] M. Knudson and K. Tumer. Coevolution of heterogeneous multi-robot teams. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Portland, OR, July 2010.

[74] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7, 1999.

[75] K. Krawiec and B. Bhanu. Coevolution and linear genetic programming for visual learning. In Cantú-Paz et al., editor, *Genetic and Evolutionary Computation — GECCO 2003*, volume 2723 of *Lecture Notes in Computer Science*, pages 332–343. Springer Berlin Heidelberg, 2003.

[76] G. E. Liepins, M. R. Hilliard, Mark Palmer, and Gita Rangarajan. Credit assignment and discovery in classifier systems. *International Journal of Intelligent Systems*, 6(1):55–69, 1991.

[77] A.B. MacKenzie and S.B. Wicker. Game theory and the design of self-configuring, adaptive wireless networks. *Communications Magazine, IEEE*, 39(11):126–131, Nov 2001.

[78] G.J. Mailath. Do people play nash equilibrium? lessons from evolutionary game theory. *Journal of Economic Literature*, 36(3):pp. 1347–1374, 1998.

[79] J. Martikainen and S.J. 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.

[80] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4:1–32, 1996.

[81] R. Miikkulainen. Evolving neural networks. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, pages 2441–2460, New York, NY, USA, 2010. ACM.

[82] L. Moreau. Stability of multiagent systems with time-dependent communication links. *Automatic Control, IEEE Transactions on*, 50(2):169–182, Feb 2005.

[83] 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.

[84] M. Nguyen-Duc, J.-P. Briot, and A. Drogoul. An application of multi-agent coordination techniques in air traffic management. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 622–625, Oct 2003.

[85] V. Oduguwa and R. Roy. Multiobjective Optimization of Rolling Rod Product Design Using Metamodeling Approach. pages 1164–1171, 2002.

[86] P. Ogren, E. Fiorelli, and N.E. Leonard. Cooperative control of mobile sensor networks:adaptive gradient climbing in a distributed environment. *Automatic Control, IEEE Transactions on*, 49(8):1292–1302, Aug 2004.

[87] I. Paenke, J. Branke, and Yaochu Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *Evolutionary Computation, IEEE Transactions on*, 10(4):405–420, Aug 2006.

[88] L. Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evolutionary Computation Journal*, 2010.

[89] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. In *Journal of Autonomous Agents and Multiagent Systems*, 2005.

[90] L. Panait, S. Luke, and R.P. Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *Evolutionary Computation, IEEE Transactions on*, 10(6):629 –645, Dec 2006.

[91] L. Panait, K. Tuyls, and S. Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *J. Mach. Learn. Res.*, 9:423–457, June 2008.

[92] L. Panait, R.P. Wiegand, and S. Luke. A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 573–584. Springer Berlin Heidelberg, 2004.

[93] 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.

[94] M. Potter and K. Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor, H. Schwefel, and R. Mnner, editors, *Parallel Problem Solving from Nature PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 249–257. Springer Berlin Heidelberg, 1994.

[95] 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.

[96] D. Ramage, D. Hall, R. Nallapati, and C.D. Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 248–256, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[97] J. Randlov and P. Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *In Proceedings of the 15th International Conference on Machine Learning*, pages 463–471. 1998.

[98] D. Ray and R. Vohra. A theory of endogenous coalition structures. *Games and Economic Behavior*, 26(2):286 – 336, 1999.

[99] M. Reyes-Sierra and C.A. Coello. A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 65–72 Vol.1, Sept 2005.

[100] C.P. Roca, J.A. Cuesta, and A. Snchez. Evolutionary game theory: Temporal and spatial effects beyond replicator dynamics. *Physics of Life Reviews*, 6(4):208 – 249, 2009.

[101] C. Rosin and R. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5:1–29, 1996.

[102] A.E. Roth. The economist as engineer: Game theory, experimentation, and computation as tools for design economics. *Econometrica*, 70(4):1341–1378, 2002.

[103] W. Sandholm. Evolutionary game theory. In R.A. Meyers, editor, *Computational Complexity*, pages 1000–1029. Springer New York, 2012.

[104] A.G. Sanfey. Social decision-making: Insights from game theory and neuroscience. *Science*, 318(5850):598–602, 2007.

[105] R. Sarkar, H.A. Abbas, and S.M.R. Karim. An evolutionary algorithm for constrained multiobjective for optimization problems, 2001.

[106] W. Schultz. Neural coding of basic reward terms of animal learning theory, game theory, microeconomics and behavioural ecology. *Current Opinion in Neurobiology*, 14(2):139 – 147, 2004.

[107] L.M. Sim, D.M. Dias, and M.A.C. Pacheco. Refinery scheduling optimization using genetic algorithms and cooperative coevolution. In *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, pages 151 –158, april 2007.

[108] J.M. Smith. Evolutionary game theory. In C. Barigozzi, editor, *Vito Volterra Symposium on Mathematical Models in Biology*, volume 39 of *Lecture Notes in Biomathematics*, pages 73–81. Springer Berlin Heidelberg, 1980.

[109] D. Sofge, K. De Jong, and A. Schultz. A blended population approach to cooperative coevolution for decomposition of complex problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 413 –418, may 2002.

[110] T. Soule and R.B. Heckendorn. A developmental approach to evolving scalable hierarchies for multi-agent swarms. In *"Proceedings of the 12th Annual Conference on Genetic and Evolutionary Compuation – GECCO-2010.* ACM, 2010.

[111] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[112] K. C. Tan, Y. H. Chew, T. H. Lee, and Y. J. Yang. A cooperative coevolutionary algorithm for multiobjective optimization. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 1, pages 390–395 vol.1, Oct 2003.

[113] K.C. Tan, Y.J. Yang, and T.H. Lee. A distributed cooperative coevolutionary algorithm for multiobjective optimization. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 4, pages 2513 – 2520 Vol.4, dec. 2003.

[114] A.C. Tenorio-Gonzalez, E.F. Morales, and L. Villaseor-Pineda. Dynamic reward shaping: Training a robot by voice. In A. Kuri-Morales and G.R. Simari, editors, *Advances in Artificial Intelligence IBERAMIA 2010*, volume 6433 of *Lecture Notes in Computer Science*, pages 483–492. Springer Berlin Heidelberg, 2010.

[115] M. Tomassini and C.S.D. Calcolo. A survey of genetic algorithms. *Annual Reviews of Computational Physics, World Scientifics*, pages 87–118, 1995.

[116] C. Tomlin, G.J. Pappas, and S. Sastry. Conflict resolution for air traffic management: a study in multiagent hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):509–521, Apr 1998.

[117] K. Tumer and A. Agogino. Multiagent learning for black box system reward functions. In *Advances in Complex Systems*. 2009.

[118] K. Tumer and A. K. Agogino. Improving air traffic management with a learning multiagent system. *Intelligent Systems*, 24(1), Jan/Feb 2009.

[119] K. Tumer and A. K. Agogino. A multiagent approach to managing air traffic flow. *Journal of Autonomous Agents and Multi-Agent Systems*, 24(1):1–25, 2012.

[120] K. Tumer and N. Khani. Learning from actions not taken in multiagent systems. *Advances in Complex Systems*, 12(4-5):455–473, 2009.

[121] K. Tuyls and K. Tumer. Multiagent learning. In G. Weiss, editor, *Multiagent Systems*. Springer, 2013.

[122] R.P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, 2003.

[123] R.P. Wiegand, W.C. Liles, and K.A. De Jong. Analyzing cooperative coevolution with evolutionary game theory. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1600 –1605, 2002.

[124] R.P. Wiegand, W.C. Liles, and K.A. Jong. The effects of representational bias on collaboration methods in cooperative coevolution. In J.J.M. Guervós, P. Adamidis, H.G. Beyer, H.P. Schwefel, and J.L. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature — PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 257–268. Springer Berlin Heidelberg, 2002.

[125] R.P. Wiegand, W.C. Liles, and K.A. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *In Proceedings from the Genetic and Evolutionary Computation Conference*, pages 1235–1242. Morgan Kaufmann, 2001.

[126] R.P. Wiegand and M.A. Potter. Robustness in cooperative coevolution. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 369–376, New York, NY, USA, 2006. ACM.

[127] S.A. Williamson, E.H. Gerding, and N.R. Jennings. Reward shaping for valuing communications during multi-agent coordination. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 641–648, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[128] L. Willmes, T. Back, Yaochu Jin, and B. Sendhoff. Comparing neural networks and kriging for fitness approximation in evolutionary optimization. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 663–670 Vol.1, Dec 2003.

[129] Z. Yang, K. Tang, and X. Yao. Multilevel cooperative coevolution for large scale optimization. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1663 –1670, june 2008.

[130] A. Zamuda, J. Brest, B. Boskovic, and V. Zumer. Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3718 – 3725, june 2008.

[131] A. Zhou, B. Qu, H. Li, S. Zhao, P. Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32 – 49, 2011.

[132] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.

[133] E. Zitzler. Evolutionary algorithms, multiobjective optimization, and applications. Technical report, Swiss Federal Institute of Technology, Zurich, 2003.