AN ABSTRACT OF THE THESIS OF

Aida Rahmattalabi for the degree of Master of Science in Robotics presented on August 2, 2016.

Title: D_{++}: Structural Credit Assignment in Tightly Coupled Multiagent Domains

Abstract approved: _____

Kagan Tumer

Autonomous multiagent teams can be used in complex exploration tasks to both expedite the exploration and improve the efficiency. However, use of multiagent systems presents additional challenges. Specifically, in domains where the agents' actions are tightly coupled, coordinating multiple agents to achieve cooperative behavior at the group level is difficult. In this work, we demonstrate that reward shaping can greatly benefit learning in tightly coupled multiagent exploration tasks. We argue that in tightly coupled domains, effective coordination depends on rewarding *stepping stone* actions, actions that would improve system's objective but are not rewarded because other agents have not yet found their proper actions. To this end, we build upon the current work in multiagent structural credit assignment literature and we extend the idea of *counterfactuals* introduced in difference evaluation functions [2]. Difference evaluation functions have a number of properties that make them ideal as learning signal, such as sensitivity to agent's actions and alignment with the global system objective. However, they fail to tackle the coordination problem in domains where the agent coupling is tight. Extending the idea of *counterfactuals*, we propose a novel reward structure, D_{++}. We investigate the performance of the D_{++} in two different multiagent domains. We show that while both global team performance and the difference evaluation function fail to properly reward the *stepping stone* actions, our proposed algorithm successfully rewards such behaviors and provides superior performance (166% performance improvement and a quadruple convergence speed up) compared to policies learned using either the global reward or the difference reward [2].

D_{++}: Structural Credit Assignment in Tightly Coupled Multiagent Domains

by

Aida Rahmattalabi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented August 2, 2016
Commencement June 2017

Master of Science thesis of Aida Rahmattalabi presented on August 2, 2016.

APPROVED:

Major Professor, representing Robotics

Head of the School of Mechanical, Industrial and Manufacturing Engineering

---

Dean of the Graduate School

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

Aida Rahmattalabi, Author

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my adviser, Professor Kagan Tumer, for supporting my research, providing guidance and advising me to the completion of this work. Without his patience and insight, none of this would have been possible. I would like to thank my committee members for their precious time and support, and the AADI laboratory for providing me with useful insight, support, and feedback throughout this process. Lastly, I would like to thank my family for their unwavering support and constant encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

Autonomous multiagent teams can be used in complex tasks, such as environment exploration for improved information gathering over single robot systems in terms of both speed and effectiveness. However, coordinating a team of agents so that they collectively achieve a common goal is a complex control problem. Especially, as the coupling between the agents increases, the mutual dependence of the agents on each others' performance [19] will grow and this poses additional challenges to the coordination task. In such cases, the performance of the multiagent system is highly sensitive to the level of coordination among the individuals' policies.

Additionally, in many multiagent problems, such as space exploration, environmental monitoring, and search and rescue tasks, only a high-level description of the task is at hand. Communication, which is a means to improve the performance of cooperative agents is also often expensive or limited. Effective team performance becomes even more challenging when robots' actions are tightly coupled requiring agents to extensively coordinate their actions in order fulfill their mission. In such cases, coordinated policies can be difficult to define a priori and distributed policy learning is often employed to optimize team strategies.

Distributed policy learning has been demonstrated to produce effective team performance in different multiagent tasks [21, 25, 40, 43, 44]. In particular, reward-shaping techniques [13, 14, 31] have been used to address the structural credit assignment problem for implicit coordination solutions where inter-agent communication is unavailable. Structural credit assignment is inherent in multiagent domains and it is the problem of determining how a single agent contributes to a system involving many other agents. For a reinforcement learner to properly learn the task, this credit assignment problem needs to be solved such that the resulting reward is both *aligned* to the system's global objective and *sensitive* to the individual agent's action. The structural credit assignment problem has been studied in numerous domains including foraging robots [29], robotic soccer [48] and network routing [47] and multi-rover domain [5]. Particularly, the difference evaluation function introduced in [2, 4] makes use of *counterfactuals* to query the

direct effect of an individual's contribution to the team performance. This reward signal has been shown to be computed from locally available information [9] and it is both *aligned* to the system's global objective and *sensitive* to the individual agent's action, and therefore provides an effective training signal when the system reward function is smooth.

However, most of the multiagent problems that have been the subject of the multiagent learning studies can be considered as loosely coupled. This is in comparison to tasks such as pushing heavy boxes that require multiple agents to collaborate for success and demands a higher level of inter-agent coordination.

In general, most multiagent tasks can be broadly categorized into [20]:

- Tasks where a single agent can accomplish, but having multiple agents expedites the process or improves the outcome. In this work, we refer to such tasks as loosely coupled. Examples of this type of task are terrain mapping or vacuum cleaning robots.

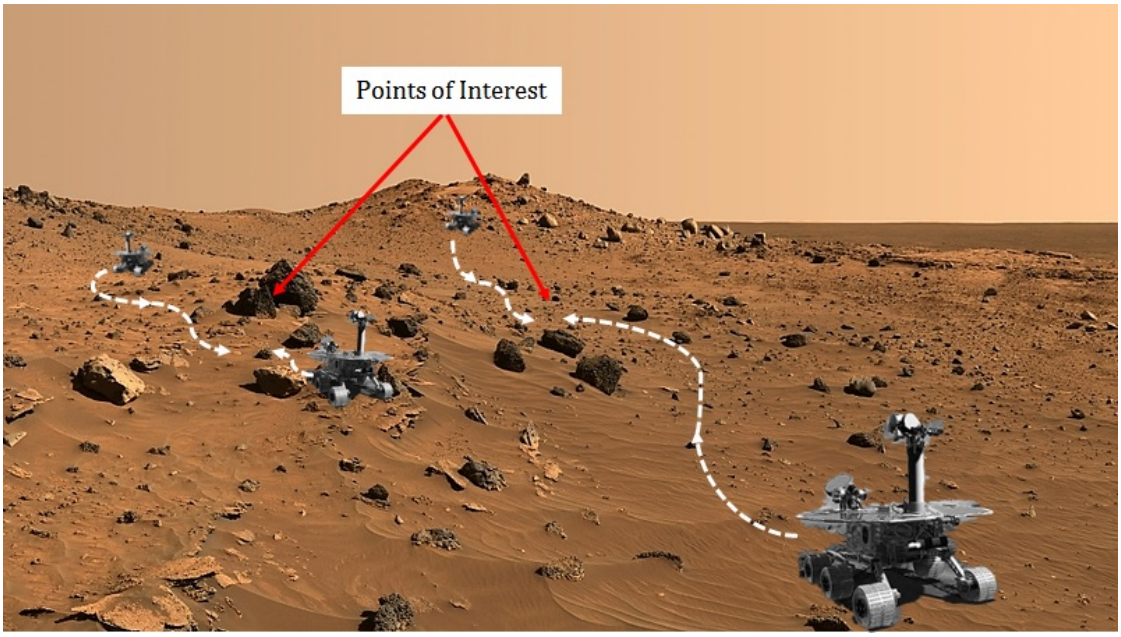
- Tasks where multiple agents are required for their success. We refer to such tasks as tightly coupled tasks. Examples of this type of task are carrying an object or pushing a heavy box where only multiple agents can manage to perform.

While in both cases, different levels of coordination are required, in the first case, a failure of coordination leads to inefficient use of resources and prolonged task accomplishment whereas in the second, it leads to a complete system failure and therefore the success of the system is highly dependent on establishing proper coordination among the agents.

From an agent learning perspective, missions that introduce tightly coupled actions consist of reward functions that are inherently non-smooth and often involve step changes that represent the specific multiagent coupling requirements of the tasks. In other words, agents do not receive any reward, which is vital to their learning, until they accomplish the task. This is one of the major problems that challenges many existing reward-shaping techniques since they are not capable of providing an adequate evaluation signal for learning joint policies necessary for achieving such tightly coupled tasks.

In this work and in order to improve policy learning for tightly coupled problems, we focus on the credit assignment in multiagent learning and we attack the reward shaping problem by proposing the notion of *stepping stone* actions. Stepping stone

actions are defined as agent actions that are aligned with the system objective and are potentially useful for the task achievement. We focus on rewarding such actions to improve coordination. To this end, we extend the idea in the difference reward and explore the manipulation of *counterfactuals* to provide agents with a stronger feedback signal on potential joint actions. D_{++}, our proposed reward function computes the effect of introducing multiple identical agents to the system and executes a targeted sweep over the number of available agents providing a rapid estimate of the joint action required to achieve a task. D_{++} is then used to compute the evaluation signal in a cooperative coevolutionary algorithm for training the neural network control policies of each agent.



http://www.teciber.com/wp-content/uploads/2015/03/marte.jpg

Figure 1.1: Rover domain representation with sample rover paths. Multiple simultaneous observations (in this case two) must be made of a particular POI to have any value to the system. Starting from different locations, well-coordinated rovers navigate to the POIs and make observations.

We demonstrate our proposed reward framework on a Cooperatively Coupled Rover Domain (CCRD), figure 1.1, which is an extension of the Continuous Rover Domain developed by [4]. In this domain, a set of rovers must coordinate their actions to collectively optimize coverage over a set of environmental points of interest (POIs). The CCRD increases the coordination complexity by requiring teams of agents to simultaneously observe each POI. Here, agents must not only optimize coverage of POIs, but

they must also form teams and the teams must coordinate within themselves to optimize coverage of their POI as well (since the proximity to the POIs influences the observation value). This is difficult because there are two different coordination problems going on concurrently. At a high level, all agents within the system must coordinate to provide an optimal coverage of POIs. Also, agents must coordinate among themselves to form observation teams, and the agents comprising the teams must coordinate their actions to optimally select and observe a given POI within its observable range (teams either observe a POI together or not at all). This tight coupling between agents both at a system level and at a team level presents a complex coordination problem.

We evaluate our method in two different monitoring tasks, one which is agnostic to the types of agents that form an observation team (*homogeneous* case), and one which requires a specific set of *heterogeneous* agent types to successfully perform the task.

We show that with D_{++}, the team of agents is able to greatly improve policy learning rates over existing shaped rewards in terms of the overall team information gathering performance and time it takes to achieve such performance. Furthermore, in cases where learning via the global system evaluation or difference reward completely fail to achieve an effective policy, the team policies learned using D_{++} are able to successfully achieve the mission objectives.

The remainder of this thesis is structured as follows. Chapter 2 provides the background on the multiagent systems and it also provides an overview of previous work and various studies in the area of learning in multiagent systems. It describes mathematical models of learning for multiagent systems and details on the learning algorithms used in the current work. Finally, we describe the structural credit assignment problem and discuss some of the properties that are important in a reward function. We then describe the difference evaluation function which forms the basis of the present work's contribution and we define the idea of *counterfactuals* introduced within that context.

In chapter 3, the D_{++} is introduced as the extension to difference evaluation function. In this section, we will provide details on how the idea of *counterfactuals* is extended in the definition of D_{++} and we discuss implementation details. We then discuss the computation complexity of D_{++} compared to both G and D.

Chapter 4 and 5 introduce two different tightly coupled problems as our test domains for studying tightly coupled multiagent learning. The first domain is devised as a simplified version of the rover domain. As mentioned earlier, the second domain is an

extension of the continuous rover domain for a tightly coupled environmental monitoring task. We also provide experimental results for different learning scenarios in both cases, such as homogeneous/heterogeneous agents and robustness analysis. Finally, chapter 6 contains the discussion and conclusions of this work. We also provide suggestions for future research directions.

Chapter 2: Background

2.1 Multiagent Systems

Multiagent systems have many benefits compared to single agent systems including scalability, robustness to component failures and environmental uncertainty, and adaptability to changing system requirements. Several successful applications of agent coordination include search and rescue, mine collection and mobile sensor network tasks. [21, 25, 44]. Comparing to the single-agent systems, multiple agents often perform the task faster and in a more efficient manner. In addition, in some cases, multiple agents are actually required to cooperate to successfully perform the task since the task is either too difficult for a single agent or outside its capabilities. For such tasks, a single robot with more functionality may be able to accomplish the task at hand, however, it consumes a great deal of time and resources to fulfill the task and it is also potentially less robust.

However, multiagent systems present additional challenges mainly because they involve multiple interacting agents which have a mutual impact on each others' performance. The problem complexity rapidly rises with the number of agents and the level of the agent-to-agent coupling. To reduce such complexity, different approaches have been investigated. For example, non-learning approaches such as auctions attempt to reduce complexity by limiting agent interaction and with bidding for tasks based on individual preferences. In this regard and for the case of tightly coupled problems, the task of box pushing has been investigated in [19]. Gerkey et al. [19] propose an algorithm based on a negotiation-style task allocation framework to automatically assign tasks to agents. Behavior-based approaches as developed in [33,34] also aim to remove the need for inter-agent communication and determine agents' fitness based on certain internal motivators that are computed based on the task. While the majority of non-learning multiagent approaches to agent coordination are in the areas of behavior-based and market-based task allocation, in both of these scenarios, simplifications are obtained by including prior knowledge of the task requirements into individual agents' preferences.

Distributed learning techniques including Markov Decision Processes for online mech-

anism design [35], developing reinforcement learning based algorithms [1, 8, 39], or devising agent-specific objective functions [4, 42] have also proved quite successful in many loosely coupled tasks [6, 32, 40, 43]. However, they also greatly suffer from the level of system coupling, and as the coupling increases the coordination becomes increasingly more difficult.

In this thesis, we are primarily interested in tightly coupled multiagent tasks and how learning techniques can benefit coordination and tackle the tight coupling between agents. We claim that one of the major barriers for coordinating tightly coupled agents is that the probability of agents collectively taking the right actions is low. This is crucially important since learning-based approaches rely on the agent's experience to learn the task, and with limited domain knowledge, agents' actions are initially random. Therefore in a tightly coupled domain, the agents receive no learning signal from the environment until they actually accomplish the task and as discussed, the probabilities of task accomplishment are very low due the agent couplings. It is noteworthy that this probability exponentially decreases as the degree of coupling among the agents increases. In other terms, as the number of required agents increases it becomes increasingly more difficult for the agents to coordinate and achieve the task at hand. To address this issue, we propose a reward structure to reward early policies that are aligned with the system objective regardless of the actions of other members of the multiagent team. The following section will be dedicated to giving an overview of our approach to solving such multiagent tasks.

2.2 Contribution of This Work

In this thesis, we define the notion of *stepping stone* actions as agent policies that are important in the ultimate task accomplishment but are not rewarded since they are not accompanied by the proper actions of their teammates. We claim that by rewarding such *stepping stone* actions, we can greatly improve the multiagent coordination in tightly coupled tasks. While most of the current literature in multiagent credit assignment in tightly coupled tasks uses reward functions that are hand-engineered and domain specific, we propose a general rewarding structure that rewards good agent policies using minimum domain knowledge. We extend the idea of *counterfactual* agents introduced in difference evaluation function, D [2], and propose D_{++}. Unlike D, in D_{++}, *counterfactual*

agents are added by each agent to inflate its capabilities and in this way the agent is able to assess its own performance even when its teammates are not present or have not fount their right actions. The way D_{++} is defined and deployed requires minimal domain knowledge and it is able to successfully reward the *stepping stone* actions by providing agent-specific feedback on agents' policies.

Finally, we present results in two multiagent simulation domains, proving the thesis presented in this work by demonstrating that the policies learned via D_{++} significantly outperform policies learned using either global team performance, G, or difference evaluation function, D, both in terms of performance and speed. We also demonstrate the efficiency of D_{++} by analyzing the computation cost and we show how it is increasingly more efficient as learning continues. Additionally, we evaluate how well D_{++} learners can react to the system failure which is crucially important specifically in tightly coupled tasks where the failure of each agent greatly impairs the system.

2.3 Agent Learning

Learning is often an important component of autonomous systems and autonomous multiagent systems are not an exception. In general, learning can be grouped into three different categories: supervised, unsupervised, and reward-based learning. Supervised learning is learning in the presence of a *teacher*, meaning that whether an agent's action is right or wrong is known to the system supervisor. Supervised learning works well for classification problems in which a set of training examples are available. However, in many complex real-world domains, dynamic interactions between agents and stochasticity in the environment make it impossible to know what the correct actions are. Unsupervised learning occurs when an agent simply learns patterns from its inputs and observations without receiving any supervision. A common example of unsupervised learning is clustering, in which potentially useful or related clusters are detected from a set of input examples [38]. Reward-based learning is often called *semi-supervised* learning since there is no explicit target function, but there are rewards which provide feedback for actions taken. In this work, we will focus on reward-based learning methods and we use both Monte Carlo policy evaluation and evolutionary algorithms for two different multiagent domains studied in this work. In the following sections, we will provide details on each of the learning algorithms that are applied in the problem domains.

2.3.1 Reinforcement Learning

Reinforcement learning concerns how agents should take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learners observe the state of the environment and have to learn based on the feedback received while interacting with the environment. In a multiagent system, an agent's environment consists of both the environment and the entire team of agents, making it an inherently non-static environment. The agents learn their actions by repeatedly modifying their behavior based on the rewards they receive while exploring the environment [41]. At each step, an agent chooses an action in its current state. The action is executed in the world and results in a change of state, and the agent receives a subsequent reward. In reinforcement learning, an agent has four key elements including a policy, reward function, value function, and optionally a model of the world. [41]. It is required for an agent to have a policy in order to act in an environment. By definition, the agent policy is the probability of taking an action a when the agent is in state s.

$$\pi(s, a) = p(a_t = a | s_t = s), \tag{2.1}$$

An agent's reward function reflects the agent's goal in a reinforcement learning problem [41]. The reward function provides an agent with a learning signal for actions taken. The rewards an agent receives are coupled with a value function in order to update the agent's policy. The value function approximates the expected reward for the agent to be in that state.

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s, a_t = a), \tag{2.2}$$

A reinforcement learning agent can use different value function formulations to update its policy based upon the rewards it receives, which differ based on the problem domain.

2.3.1.1 Monte Carlo Policy Evaluation

Monte Carlo (MC) policy evaluation is a simple learning method that can be used to estimate value functions. Monte Carlo methods require only experience sequences of states, actions, and rewards from on-line or simulated interactions in an environment

and it is based on averaging the sample returns. The learning algorithm is explained below:

Algorithm 1 Monte Carlo Policy Evaluation

1: Initialize
   π: the agent's policy
   $V(s)$: an arbitrary value function
2: Repeat forever:
   Generate an episode using π
   For each state observed, update the value function
   $V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$

In the above algorithm, α is the learning rate, s_t consists of the state-action of the agent and R_t is the reward received at time t. Finally, $V(s_t)$ is the approximation of the value function. In section 4, we will explain why and how we use the MC method for our problem domain and we will provide more implementation details.

2.3.2 Neural Networks

Known as universal approximators [24], neural networks (NN) have been used in many applications including classification, controls and nonlinear signal-processing [7, 22, 28]. NNs are a powerful mathematical tool to represent both discontinuous and continuous functions to arbitrary accuracy [24]. Each neural network maps a set of inputs to a set of outputs. A feed-forward neural network consists of a set of input neurons, output neurons, and hidden layer of neurons which are connected via weights, figure 2.1. Each neuron also has an activation function, which is typically a nonlinear function [30]. The weighted sum of the inputs is forwarded through the network to generate a set of outputs. As mentioned in section 2.3, there are supervised, unsupervised and reward-based methods to train neural networks. Supervised learning techniques are suitable for cases where the neural network controller has a teacher that knows the correct mapping from a set of inputs to a set of corresponding outputs, and therefore can provide the neural network with constructive feedback on actions taken, by calculating the error in the output neurons. By back-propagating the error to the network, one can adjust the weights and improve the neural network's performance over time. Unsupervised learning approaches, on the other hand, are used in cases where a neural network attempts

to cluster a set of unlabeled data using some similarity metric. Finally, reward-based learning with neural networks utilizes reward feedback based upon the network output in order to update the neural networks weights [12,22]. In the present work, each agent uses one-hidden-layer feed-forward neural networks as its controller. And since the correct policies are not known, we take a reward-based training approach that will be explained in the following sections.

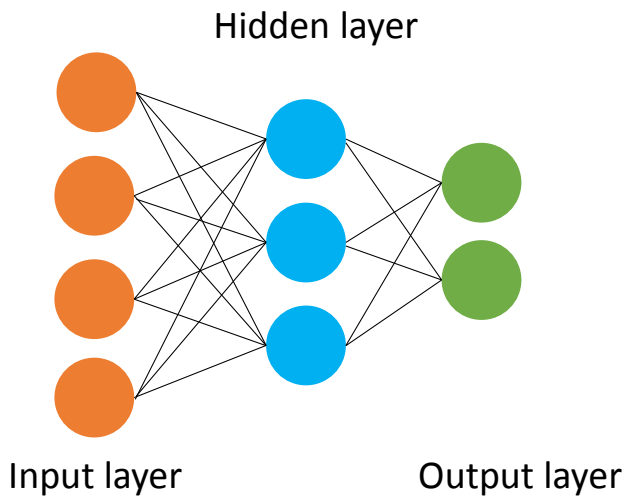


Figure 2.1: Diagram for a two layer feed-forward Neural Network (NN). The network's input, hidden and output variables are represented by neurons, and the weights are shown by links that connect neurons. In a feed-forward NN the information flows through the network from input to hidden and finally to the output layer.

2.3.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of stochastic search algorithm inspired by biological evolution, that often outperform classical optimization algorithms [18]. These algorithms have been proved successful in many single-agent and multiagent control problems [15, 27, 45]. Typically, EAs contain three basic mechanisms: solution generation, mutation, and selection. Starting with an initial set of candidate solutions (the population), these mechanisms are used to generate new solutions and retain existing

solutions that show improvement based on a system utility. Simple EAs can be applied to a variety of single-agent learning tasks, however, when dealing with large cooperative multiagent problems, modifications are required for an effective performance. One such modification is coevolution, where multiple populations evolve simultaneously in order to develop policies for interacting agents. Before explaining how coevolution works, we dedicate a section to describing how neural network controllers and evolutionary algorithms are combined together to solve the control problem in the present work.

2.3.3.1 Neuro Evolutionary algorithms

Neuro-evolutionary algorithms are a subset of evolutionary algorithms that evolve neural network policies. These algorithms evolve a pool of neural networks, representing agent policies, that is randomly created. A policy is selected based on an ϵ-greedy selection mechanism, then the selected neural network is mutated by adding a random number from *Cauchy* distribution to its weights. Then the quality of this policy is evaluated based on a fitness function and it is compared to the rest of the policies. The worst policy in the pool is removed and the cycle continues until a convergence criterion is met [17, 32].

Neuro-evolutionary methods have been shown to work well in many multiagent domains involving continuous state spaces and complex agent interactions such as the continuous rover domain in this work [4].

2.3.4 Cooperative Coevolutionary Algorithms

The standard Cooperative Coevolutionary Algorithm (CCEA) [37] has been used in this study as a base algorithm for learning control policies. CCEAs are an extension of EAs for multiagent systems and have been shown to perform well in cooperative multiagent domains [16]. The standard CCEA is detailed in Algorithm 2. In CCEA, N co-evolving populations of neural networks (neuro-controllers) are utilized to form teams comprised of N agents. One member of each population is extracted for each agent to form a team which then operates in the problem domain. At each generation, k mutated networks are generated in each population by mutating the parent networks. Then, $2k$ teams of agents are formed and simulated. The performance of each simulated team is subsequently

evaluated using a fitness function, $F(z)$, and assigned to every agent in the team. In this paper, we propose D_{++} as the fitness function, and we compare it to using G and D. Finally, k networks from each population are selected based on ϵ-greedy selection to proceed to the next generation. This process is repeated for a set of generations.

Algorithm 2 Standard CCEA

```
1: Initialize N populations of k neural networks
2: for Generation do
3:     for Population do
4:         produce k successor solutions
5:         mutate successor solutions
6:     for i = 1 → 2k do
7:         randomly select one agent from each population without replacing it into the
   population pool
8:         add agents to team T_i
9:         simulate T_i in domain
10:        assign fitness to each agent in T_i using F(z)
11:    for Population do
12:        select k solutions using ε-greedy selection
```

2.4 Structural Credit Assignment

In many multiagent coordination domains, there is a difference between maximizing the system fitness function and maximizing a single agent's fitness value. The main problem is how to provide agents with a reward signal that is more sensitive to their own actions while still reflecting the system's global objective. The problem can be referred to as structural credit assignment and is aimed at shaping the agent's reward such that it enhances its learning. In this context, Hoen and De Jong shaped the utilities of the agents such that an agent maximizing its individual utility would act to also increase the system evaluation function [36]. This work is similar to that of Agogino and Tumer, and Knudson and Tumer, who utilized difference evaluations as fitness functions to improve coordination in multiagent systems [2,26]. The difference evaluation function is a shaped reward that uses *counterfactual* agents to provide agent-specific rewards. The following section will first define the two major properties of a good learning signal and finally in section 2.4.2 we will elaborate on the definition of the difference evaluation function and

how these properties are reflected in its definition, which will also provide the necessary background for the proposed reward structure, D_{++}.

2.4.1 Factoredness and Learnability

In the context of agent policy evaluation functions used in this work, two metrics have been introduced by [2] for determining the quality of a policy evaluation function. Ideally, an agent's policy evaluation function should be able to determine:

- How its action impacted the overall system performance
- How its action impacted the evaluation it received

Feedback on how an agent impacted the system performance allows that agent to make decisions that are aligned with the global system objective. Providing agents with feedback on how their individual action impacted the evaluation they received allows agents to change their own actions in order to benefit both themselves and the system. The first property has been defined as the degree of factoredness or *alignment* between the agent policy evaluation function and the system objective, G. Detailed formulation is presented in [46]. A high degree of factoredness means that agents improving their own policy evaluation function are simultaneously improving the system performance, whereas actions that harm an agent's policy evaluation function are also harmful to the system. The second property is defined as learnability or *sensitivity*, which is the degree to which an agent's policy evaluation function is sensitive to the agent's actions as opposed to the actions of other agents. The learnability of a policy evaluation function is also defined rigorously in [46]. The learnability provides a ratio of the portion of the agents evaluation that depends upon its own actions, known as signal, and portion of its evaluation signal that depended upon all other agents' actions which are considered as noise.

2.4.2 Difference Evaluation Function

The difference evaluation function [2] is a shaped reward signal that provides agent-specific evaluation by removing a large amount of the noise created by the actions of

other agents in the system [10]. It is defined as:

$$D_{\mathbf{i}} = G\left(\mathbf{z}\right) - G\left(\mathbf{z}_{-i} \cup \mathbf{c}_i\right), \tag{2.3}$$

where $\mathbf{z}$ is the joint state, $G(\mathbf{z})$ is the global system performance, $\mathbf{z}_{-i}$ are all the state-actions on which agent i has no effect and $\mathbf{c}_i$ is the *counterfactual* term, which is a fixed vector used to replace the effects of agent i. Thus, $G(\mathbf{z}_{-i} \cup \mathbf{c}_i)$ is the evaluation of a theoretical system without the contribution of agent i. Any action taken by agent i to increase D_i simultaneously increases G. This property is defined as factoredness in the previous section, and is a key property of any shaped reward. Further, the second term in (2.3) removes portions of $G(\mathbf{z})$ that are not related to agent i, resulting in an improved signal-to-noise ratio. In other terms, agent i's impact on D_i is much higher than its relative impact on $G\left(\mathbf{z}\right)$ [46]. This is the sensitivity property as explained in section 2.4.1. Both [4], [11] have previously shown that the factoredness and sensitivity properties of the difference evaluation function result in agent-specific feedback, which leads to superior learning performance. Intuitively, the difference reward provides each agent with a reward that is proportional to its own contribution to the system performance.

Chapter 3: D_{++}: Rewards for Potentially Good Actions

One of the challenges in distributed learning is that with no/limited prior knowledge of the task and limited communication capabilities, agents must randomly search the state-action space until they receive a reward signal to evaluate their policies. It is particularly more challenging in tightly coupled domains since the task cannot be accomplished unless tight coordination among the agents is established and maintained while considering the huge joint state-action space of a multiagent system, the probability of agents simultaneously executing the right actions is very low. Thus, agents receive no feedback on their policies in large portions of the joint space. In such domains, it is critical to provide rewards for *stepping stone* actions, actions that would lead to good outcomes if joined by other agents.

3.1 Counterfactuals for Potential Actions

To illustrate, consider an exploration scenario where three rovers are required to simultaneously observe a POI. The rovers start from random locations and should synchronize their movements to reach the POIs simultaneously. However, it is unlikely that randomly moving rovers are able to learn this task within a reasonable amount of time since only after all three rovers successfully observe a POI do they receive a reward. In the absence of system feedback, they are not capable of evaluating any potential improvement in their policies or distinguishing what policies would benefit the task accomplishment in the long term.

Clearly, being able to distinguish between a case where there are two rovers close enough to a POI, waiting for one more rover to complete their subteam and a case where all three rovers are randomly moving in the world can greatly influence the learning performance.

A good reward function should enable agents to evaluate their actions in the absence of their teammates. This can lead to a huge reduction in the search space and an increase in the probability of success. Building upon the above argument, we extend the idea

of *counterfactuals* in the difference reward and we propose a reward function, referred to as D_{++}, to address the present coordination problem. The *counterfactual* that we propose computes the effect of introducing multiple agents to the system and compares it to the current state of the system. Here, let us first consider a case where only one *counterfactual* agent is added. D_{++} for one *counterfactual* agent is computed using the equation below:

$$D^1_{++}(i) = G(\mathbf{z}_{+\mathbf{i}}) - G(\mathbf{z}), \tag{3.1}$$

In equation 3.1, the agent will evaluate what the team performance would have been if there were two agents performing the task, instead of one and therefore it can evaluate whether there is any potential to score a reward with its current policy.

We can further extend the above formulation to n *counterfactual* agents. The mathematical formulation for this case can be expressed as:

$$D^n_{++}(i) = \frac{G\left(\mathbf{z}_{+(\cup_{\mathbf{i}=\mathbf{1},\ldots,\mathbf{n}})\mathbf{i}}\right) - G(\mathbf{z})}{n}, \tag{3.2}$$

In the above equation, $\mathbf{z}_{+(\cup_{\mathbf{i}=\mathbf{1},\ldots,\mathbf{n}})\mathbf{i}}$ indicates all the states on which agent i has added n *counterfactual* agents. This allows the primary agent to investigate the effect of hypothetically introducing n other agents. Also, in the formulation of D^n_{++}, if the value of n is set to -1, D^n_{++} will yield the definition of D as described in equation (2.3) noting that the *counterfactual* term $\mathbf{c}_i$ is an arbitrary vector which can indeed be set to $\vec{0}$. Also the division by n is to normalize with respect to the number of *counterfactual* agents.

By adding *counterfactual* agents, an agent inflates its capabilities and this way it is able to assess its own performance even when its teammates are not present or have not found their proper actions yet. As defined in equation 3.2, in order to calculate D_{++}, an agent needs to know about the number of required agents for a task so that it can decide on the number of *counterfactual* agents. Since we assume no prior knowledge on the number of agents required to fulfill the task, the agents should execute a sweep over the number of *counterfactual* agents.

Regarding the search computation, we make a basic assumption that in any state of the agent, the entire multiagent system should be able to accomplish the task (if at all possible). For example in the case of an exploration task, if a rover is within the

observable region of a POI, by adding the entire team of agents as *counterfactual* agents, the system can make an observation of that POI. This simple assumption is important in two ways. First, it gives us an upper bound for the number of allowable *counterfactual* agents. Second, it helps us to evaluate whether there exists a solution before we begin searching over the numbers of *counterfactual* agents in D_{++} algorithm and in this way we can greatly improve the search computation.

The search space also depends on the heterogeneity of agents as well as the heterogeneity of their actions. In the next section, first we discuss an algorithm that searches over different numbers of *counterfactual* agents, assuming that all the agents are homogeneous. Next, in section 3.3, we discuss the modification needed to address agent heterogeneity. As for the action heterogeneity, it should be noticed that the search space increases rapidly if the agent actions are heterogeneous, requiring a search over the joint action space at each state. This type of heterogeneity necessitates some knowledge about the requirements of the task to help guide the search.

To finalize the reward calculation, there is another subtlety that has to be taken into account. And that is with the difference reward, agents optimize the global system objective by calculating each agents' contribution to the system performance. This is calculated based on the difference an agent makes by executing a certain action. D_{++} further extends this idea by evaluating the impact of introducing multiple agents. In tightly coupled tasks, D does not provide any feedback unless tight coordination among the agents is established and maintained. On the other hand, D_{++} fails to provide this gradient information where sufficient number of agents have coordinated. So it is reasonable to leverage both D and D_{++} to the benefit of the system. This leads us to the final definition of our reward framework which is based on the idea of both removing and adding agents, described by D and D_{++}, respectively.

At this stage, we need a criterion to choose between D^n_{++} for different n values and D. For the selection, it is reasonable to choose the highest reward value since it corresponds to where the agent has the most impact. Accordingly, at each policy evaluation stage, both D and D^n_{++} are calculated and while incrementing n, D^n_{++} values are compared to D and whenever a value is found that is greater than D the algorithms exists and returns the highest value.

3.2 D_{++} for Homogeneous Agents

In cases where the entire team is made of agents of identical construction (homogeneous agents), the tasks are limited to redundant observations of an environment to provide robustness, or mechanical tasks that require multiple individuals to provide enough effort. In this case and in order to calculate D_{++}, each agent adds *counterfactual* agents that are identical to the agent itself, executing the same policy. The agent keeps incrementing the number of *counterfactual* agents until it reaches a value greater than D. Algorithm 3 summarizes all the steps to calculate the final reward value.

Algorithm 3 D_{++} (Homogeneous Agents)

1: calculate D_{++}^{-1} using (3.2)
2: calculate $D_{++}^{totalAgents-1}$ using (3.2)
3: **if** $D_{++}^{totalAgents-1} \leq D^{-1}$ **then**
4: Return D_{++}^{-1}
5: **else**
6: $n \leftarrow 0$
7: **repeat**
8: $n = n + 1$
9: Calculate D_{++}^{n} using (3.2)
10: **if** $D_{++}^{n} > D_{++}^{n-1}$ **then**
11: Return D_{++}^{n}
12: **until** $(n \leq totalAgents - 1)$
13: Return D_{++}^{-1}

Algorithm 3 begins with the calculation of $D = D_{++}^{-1}$ and $D_{++}^{totalAgents-1}$. Note that the $totalAgents - 1$ plus the agent itself make up the entire team of agents. If $D_{++}^{totalAgents-1}$ is less or equal to D, it means that adding agents does not benefit the agent. In other terms, it does not receive a higher reward from the system if there were more agents helping it in performing the task. In our exploration domain, this corresponds to either a case where an agent is outside the observable region of all the POIs and thus it is not possible to observe any of the POIs or a case where there are sufficient number of agents within the observable range. In the second case, D can simply be used to reward each agent based on its contribution. It can be seen that by making this assumption, we can greatly reduce the computation by eliminating the need to loop

over all the values of n before realizing all of this computation is in vain.

3.3 D_{++} for Heterogeneous Agents

So far, we have only considered agents that are homogeneous. However, some tasks may require multiple agents of differing construction and capabilities to partner with one another in order to perform a task. Using heterogeneous agents in a multiagent domain adds a great deal of potential and power at the price of added complexity. In such domains, team formation is indeed more challenging since each agent must both identify the subset of agents that it needs to collaborate with and to synchronize its action with its teammates in order to fulfill the task. In this section, we extend the D_{++} algorithm to handle coordination of heterogeneous agents.

Let us assume there are t types of agents, and in order to successfully accomplish a certain task, at least a minimum number of agents of each type should partner with each other. As an example, we refer to the exploration domain where in order for an observation to be counted as successful, n_i agents of each type i should simultaneously observe the POI within its observation radius. That is, a total of $\sum_{i=1}^{t} n_i$ agents must be present in the observable region surrounding each POI.

Comparing to the homogeneous case described before, it is clear that coordination is more challenging since the probability that the sufficient number of agents of the exact required types select the right actions is even further reduced. Thus, we need a coordination mechanism that is able to provide the useful incentive for each agent and to guide them toward the more promising policies even in the absence of their teammates. In order for D_{++} to account for such heterogeneity, we need to slightly modify the search procedure. Here we assume that the number of different types of agents as well as the number of agents of each type are known to all agents. We start by assigning $\vec{n} = \vec{0}$, where $\vec{n}$ is a vector of size t, equal to the number of types of agents. $\vec{n}$ holds the number of *counterfactual* agents needed to be added for each type in order to receive a value greater than the difference reward. In order to find $\vec{n}$, we add one agent of each type at each step, and we continue incrementing the number of agents of all types until we reach a value greater than D. Algorithm 4 summarizes the approach.

Algorithm 4 D_{++} (Heterogeneous Agents)

calculate D_{++}^{-1} using (3.2)
calculate $D_{++}^{totalAgents-1}$ using (3.2)
$\vec{n} \leftarrow \vec{0}$
if $D_{++}^{totalAgents-1} \leq D^{-1}$ **then**
    Return D_{++}^{-1}
else
    repeat
        for $t = 1 : totalTypes$ **do**
            $n_t = n_t + 1$
            Calculate D_{++}^{n} using (3.2)
            if $D_{++}^{n} > D_{++}^{n-1}$ **then**
                Return D_{++}^{n}
    until $(n \leq totalAgents - 1)$
Return D_{++}^{-1}

3.4 Computational Complexity

In computing the difference reward, two calculations of the global evaluation function are made by each agent ($G(\mathbf{z})$ and $G(\mathbf{z}_{-i} \cup \mathbf{c}_i)$) while an agent learning with the global reward only calls $G(\mathbf{z})$ once for each policy evaluation. Here, the assumption is that each agent makes the reward computations locally and no feedback is broadcast to the system. For D_{++}, however, this number is increased since agents must continue adding *counterfactual* agents until they reach a value greater than D, and each iteration requires additional calls to $G(\mathbf{z})$. This adds to the computational time of the learning algorithm. Table 3.1 compares the computational complexity for evaluating a single policy as the number of calls to $G(\mathbf{z})$.

Table 3.1: Comparison of number of calls to G for three different reward functions

Evaluation function	G	D	D_{++}
Calls to G	1	2	$c_i + 3$

In Table 3.1, c_i is the minimum number of *counterfactual* agents needed to be added

by agent i. This term is bounded between $[0, \textit{total number of agents} - 1]$ since we assume the entire team of agents should be able to accomplish the task. In cases where a sufficient number of agents are present to complete a coupled task, c_i is equal to 0 (i.e. D provides a sufficient learning signal). In cases where adding any number of agents cannot increase the system performance, c_i is also 0. This is because, when calculating the D_{++} reward, first $(\textit{total number of agents} - 1)$ agents are added and if the returning reward is not greater than D, it can be understood that iterating over fewer *counterfactual* agents will similarly not yield a better reward. Lastly, if not enough agents are within the observable radius of a POI, c_i takes a value between $[1, \textit{required number of observations} - 1]$. We will revisit this problem in the results section where we show that the average value of c_i decreases for the team of agents throughout learning, making it increasingly more efficient to use D_{++} as the learning signal. Also, when studying the computation complexity, one of the immediate concerns is the scalability of the algorithm. Similar to D, D_{++} also scales linearly with respect to the number of agents which makes it scalable to larger domains.

Chapter 4: Cooperatively Coupled Stateless Rover Domain

In many real world applications such as search and rescue, and exploration and data gathering [3, 23], using a team of agents greatly decreases the amount of time required to complete tasks and significantly improves the overall performance. The rover domain is a prime example of a system where multiple autonomous agents need to coordinate their actions in order to collectively optimize the system performance. In this domain, a team of rovers is looking for a set of Points of Interest (POIs) to observe, explore or rescue depending on the application. Every POI has a value associated with it. Also, the tight coupling of this domain originates from the fact that each POI requires multiple simultaneous observations to be successfully observed. In this work and in order to investigate the performance of the proposed reward function D_{++}, we defined two different test domains based on the cooperatively coupled rover domain. In this chapter, we will explain the Cooperatively Coupled Stateless Rover Domain (CCSRD) and in chapter 5, we will describe the more complicated version of the tightly coupled rover domain that includes continuous states and actions. The goal of both domains is to collectively observe as many environmental POIs as possible. Specifically, we investigate the interactions and coordination between the rovers when multiple rovers are required to observe a POI.

The Stateless Cooperative Rover Domain is a simplified version of the actual cooperative rover domain which is the main subject of this work. The stateless domain is a benchmark arena that allows us to rapidly test and compare different policy evaluation functions without many of the external confounding factors that can impact learning.

Over time, the agents can estimate the expected reward of each policy by selecting that policy for a number of times and observing the outcome. As mentioned, in this domain, each agent has no state (or we can define it as a single state problem) and can only select from a fixed set of actions which correspond to different POIs in the system. However, in order to introduce the tight coupling to the system, we require each POI to be observed by a certain number of agents and below this required number of observations, agents will not receive credit for observing that POI. Finally, we use the

MC policy evaluation method described in algorithm 1 for learning coordinated policies. The policy selection strategy is ϵ-greedy meaning that the algorithm chooses the action with the highest value with probability $(1 - \epsilon)$ and with probability ϵ will choose a random action.

4.1 Objective Functions

The global utility of the system is the sum of the observation values of all the POIs, formally expressed bellow.

$$G(\mathbf{z}) = \sum_{n=1}^{N} V_n(z), \tag{4.1}$$

In equation 4.1, $V_n(z)$ is the value of observing the n-th POI which is a function of the number of observations for that POI. In other terms, the number of agents that have selected the POI. In a tightly coupled domain, this function is defined as a step function as below, in which v is a fixed value and C is the number of required observations for each POI (Local Reward).

$$V_n(z) = \begin{cases} v, & \text{if } numberofobservations \geq C \\ 0, & \text{otherwise} \end{cases}$$

Generally, different objective functions can be defined depending on the task requirements. Figure 4.1 shows three different functions representing the observation value for a POI. In figure 4.1, the left function indicates an smooth transition from zero to the ultimate reward that can be received for observing a POI. However, at C (C=5 in this case), this reward becomes constant since there is no added value for redundant agents observing the same POI. Another function studied in this work, is linear (see figure 4.1.b). Like the smooth function, here the agents still receive a reward for observing a POI even if they are not sufficient to fully accomplish the task. And finally, the step function strictly requires C number of agents to choose the same POI in order to receive a reward.

While the main focus of the present work is to address the multiagent coordination in tightly coupled tasks which is equivalent to the step objective function in figure 4.1,

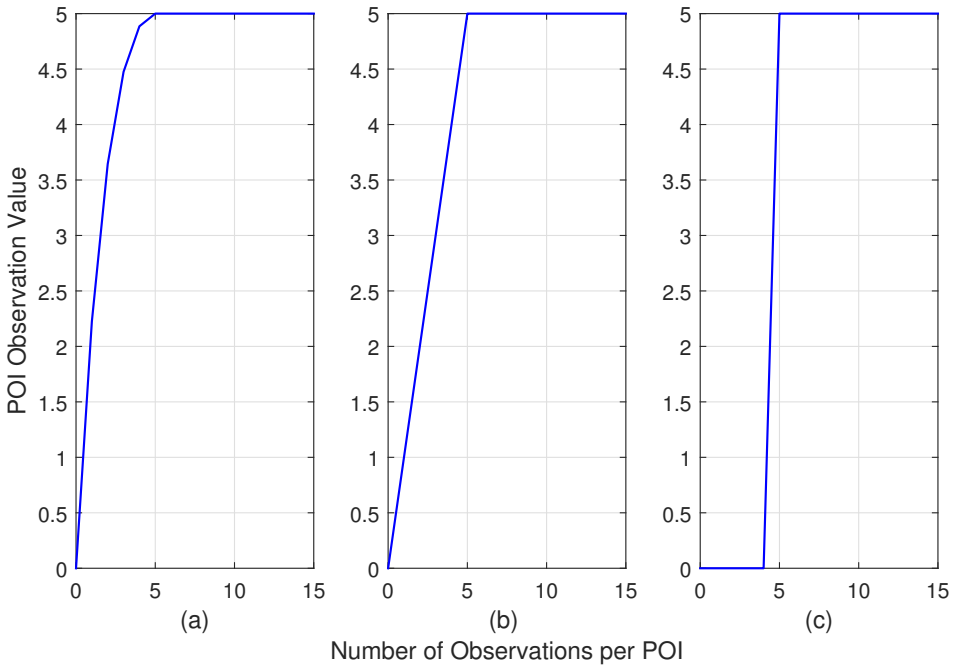


Figure 4.1: Different objective functions for observing a single POI. The first type is a smooth function providing an observation value even when fewer observations than the required threshold are executed, the second type is a linear functions and the third objective is an step that strictly requires a minimum number of observations to be made to account as success.

we also study the linear and smooth objective functions and we show that how the choice of objective function can greatly change the expected behavior from the agents. In the following sections, we show that the tightly coupled task is where D_{++} is the most beneficial to use as the learning signal compared to D, whereas for both linear and smooth objective functions shown in figure 4.1, D and D_{++} result in very close learning performance. More discussion will be provide in sections 4.3.2 and 4.3.3.

4.2 Experimental Setup

To investigate the performance of the proposed shaped reward, different sets of experiments have been performed. We use both G and D as the baseline rewards and we compare the performance of D_{++} learners against those. We initialize the CCSRD domain with 40 agents and 25 POIs. The number of POIs is intentionally set high to make it harder for multiple agents to stumble upon one POI. It also allows us to investigate how well D_{++} learners perform if the number of actions increases. It is clear that by increasing the number of actions, the joint-action space of the team of agents exponentially increases, making the coordination more difficult. We show the results for the different objectives shown in figure 4.1. We test the D_{++} performance for different numbers of required observations. Here the results are averaged over 50 statistical runs.

4.3 Results

4.3.1 Objective: Step Function

Figure 4.2, indicates the learning performance with respect to the number of calls to G. These results show how the agents learn policies using either G, D or D_{++} as the learning signal. As you can see, D_{++} learners significantly learn faster than G and D learners. This can be explained in terms of the *stepping stone* actions. D_{++} manages to reward good agent policies early in the learning process, preventing agents from continuing their random behaviors. This hugely boosts the performance and expedites learning (6 times faster learning compared to D). Also noteworthy is that these results are compared against equal number of calls to G which takes into account the fact that D_{++} requires more calls to G in order to find the right number of *counterfactual* agents.

In figure 4.3, we investigate the impact of different numbers of required observations on the ultimate performance of policies learned using the three reward signals. The y axis represents the performance scaled by the best achievable performance in this domain.

It is expected that by increasing the observation threshold, both G and D quickly start to perform poorly. Reason is the the fact that by increasing the observation threshold, we are requiring more and more agents to stumble on the same POI and the probability of this event is really low at higher threshold values. Ultimately at $C = 10$, both G and D result in zero performance while D_{++} yields high-reward policies and shows

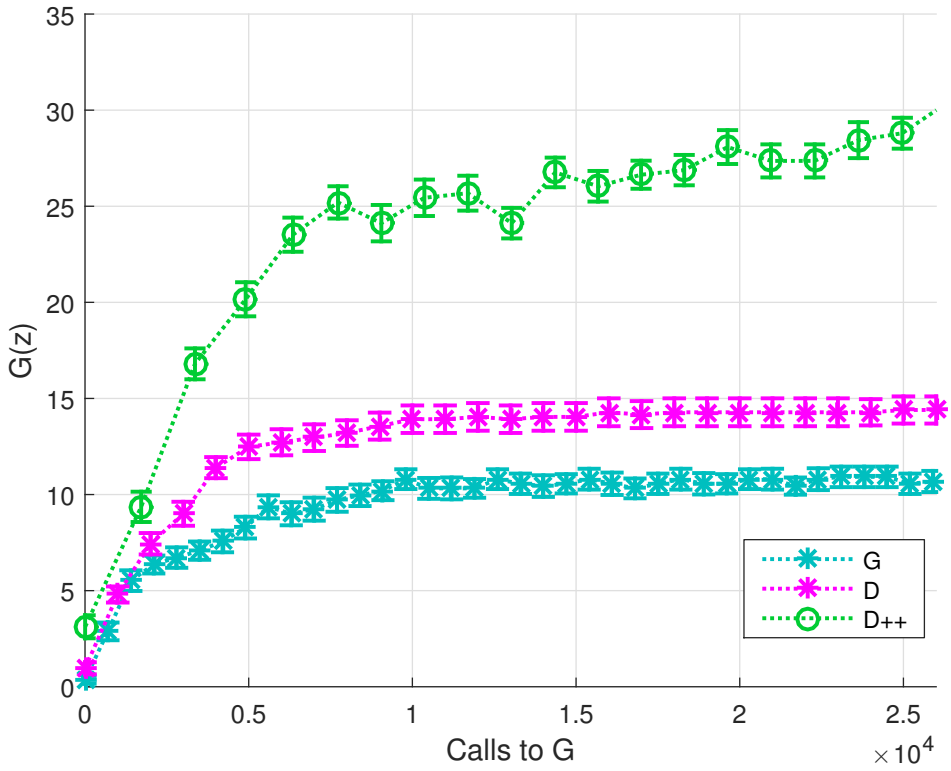


Figure 4.2: Observation performance of policies trained on G, D, D_{++} for 40 agents, 25 POIs, 6 required observations for the step objective function

less sensitivity to the observation threshold. Also, as you can see, at $C = 2$, D performs slightly better than D_{++}. To explain this, consider a scenario where one agent has chosen a POI and needs one other agent to accomplish the observation. In this case and according to the formulation of D_{++} (see equation 3.1), by adding one *counerfactual* agents, the agent receives a value that is equal to the reward it receives if it had actually achieved the task. Therefore, in both of these cases, the agent is rewarded equally which impedes learning the right action. However, you should note that the policies learned using D_{++} still outperform those learned using G and the reason explained above does not have the same impact in higher observation thresholds.

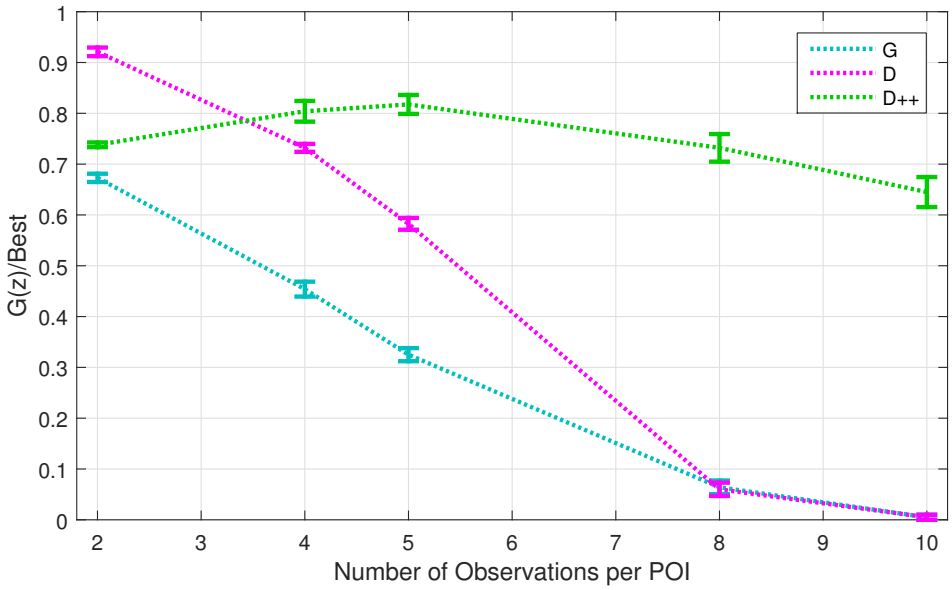


Figure 4.3: Final Observation performance of policies trained on G, D, D_{++} for 40 agents, 25 POIs with respect to different observation thresholds for the step objective function.

4.3.2 Objective: Linear Function

In figure 4.4, you can see the final learning performance for G, D and D_{++} learners with an objective function that linearly rewards them for observing a POI based on the number of present agents. In this case, even if an insufficient number of agents observe a POI, they still receive a reward. An example of this can be the case where having more agents taking scans of a POI provides more useful data but this is until we have enough agents giving a full coverage of the POI and no more information will be added by having redundant agents.

As you can see, both D and D_{++}, perform very closely to the optimal performance and keep this high performance as the observation threshold increases. Part of the reason is that initially the agents randomly choose POIs and this in fact is close the optimal behavior expected for this type of objective function. Another reason is the power of both D and D_{++} in providing agent-specific reward signals that change with respect to agents' own actions. These results also show that if the objective function is continuous, D_{++} performs very closely to D. This is because in both D and D_{++}, agents are looking for a gradient in the the objective function and if objective function is a relatively smooth

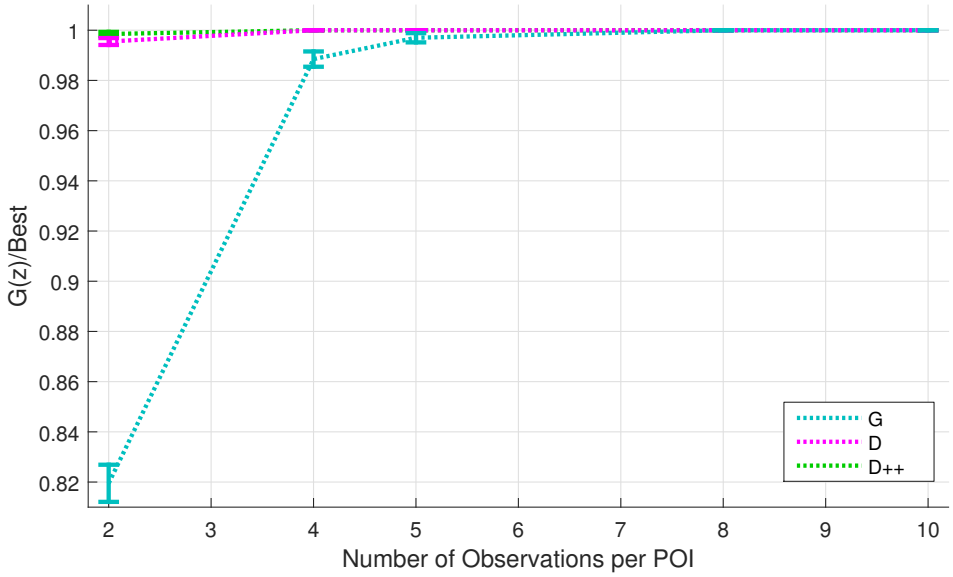


Figure 4.4: Final Observation performance of policies trained on G, D, D_{++} for 40 agents, 25 POIs with respect to different observation thresholds for the linear objective function.

function (not step-like), D can provide the gradient feedback needed to evaluate policies. Thus D and D_{++} do not differ greatly in their learning performance.

4.3.3 Objective: Smooth Function

In this section, we present the result for the final performance of G, D and D_{++} learners, learning the smooth function for observing the POI.

Likewise, both D and D_{++} learn the optimal behavior for any value of observation threshold. Whereas, it is harder for G to gain such optimal behavior. Comparing to the results presented in 4.4, G even performs slightly worse. We explain this by pointing to the types of the functions agents should learn. In the previous section, the function was piece-wise linear whereas the current function is composed of both nonlinear and linear functions which makes the learning more challenging. However, because of the properties of both D and D_{++}, they are able to extract a signal that is easier for them to learn compered to the actual nonlinear function. Finally, similar to the linear objective function, here D and D_{++} perform very closely and the reason is that they are both based on the objective function's gradient which returns similar values if the objective function

is continuous, leading to almost equal learning performance.

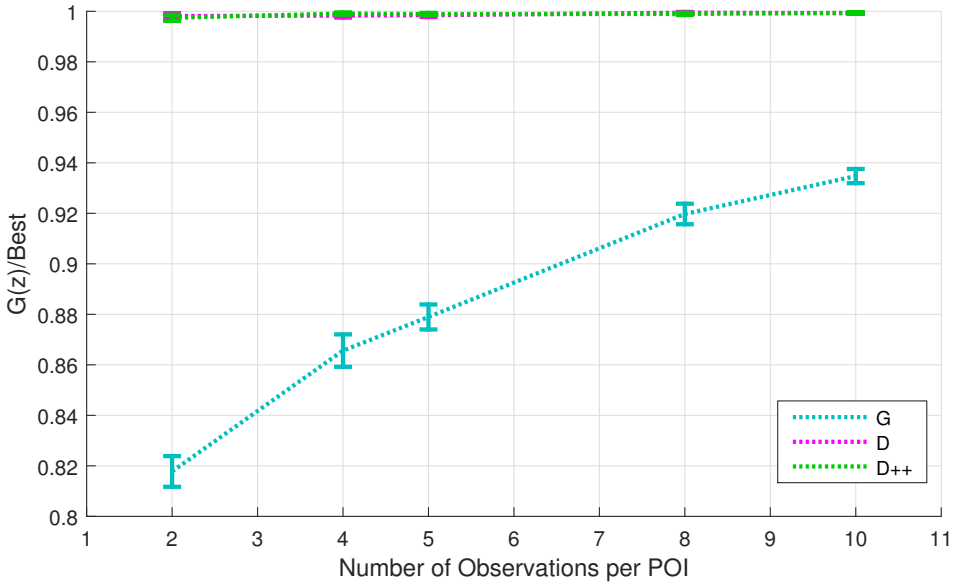


Figure 4.5: Final Observation performance of policies trained on G, D, D_{++} for 40 agents, 25 POIs with respect to different observation thresholds for the smooth function objective.

Chapter 5: Cooperatively Coupled Continuous Rover Domain

In this section, we discuss the Cooperatively Coupled Rover Domain that has been adapted and modified from the original Continuous Rover Domain [4]. This domain contains a set of rovers (agents) that are able to move around in a two-dimensional plane to observe Points of Interest (POIs). Unlike the Stateless Cooperative Rover Domain explained in section 4, each rover has a state that is determined by two types of sensors that recognize POIs and other Rovers. Each rover has a total of 8 sensors (four of each type), similar to the original continuous rover domain as described in [5]. These sensors return the density of either POIs or other rovers in this domain, which serves as the state input of the rovers' controllers.

Each POI has an observation radius which is the maximum distance from which a rover can observe the POI. The goal of these rovers is to collectively observe as many environmental POIs as possible. The key difficulty of this domain is that multiple simultaneous observations of a POI are required otherwise no reward is received by the team for those observations. A POI is considered observed only if $m > C$ rovers observe a POI from within a specified but unknown sensing distance (C is the number of required observations). If fewer than m rovers observe the POI, or if m rovers are not in the observation distance, no credit is obtained for those observations. This problem formulation ensures that team coordination is essential to the completion of the task. In this domain, it is impractical for an individual rover to search the entire environment. Instead, the rovers must coordinate in order to divide up the coverage areas and to maximize the number of POIs found while still forming teams of observations for each POI. This requirement that multiple rovers are necessary to observe each POI leads to the Cooperatively Coupled Rover Domain used in this work.

Each POI in this domain has a value V_i assigned to it. The goal of the agents is to form teams and for the teams to optimize the observation value of the environmental POIs. The value of an observation is dependent of the proximity to the POI and also the value assigned to it. Figure 5.1 demonstrates the Continuous Cooperative Rover Domain.

5.1 Agent State Representation

As mentioned in the previous section, in the Continuous Cooperative Rover Domain, each agent has two types of sensors (POI, Rover) and a total of 8 sensors (four of each type). Each rover's view is divided into four quadrants and each quadrant contains both a POI and a rover sensor. The quadrant axes are oriented along the rover's heading at every given time-step t (see Fig. 5.1). Each type of sensor returns a value representing the density of POIs or rovers (respectively) in that quadrant. The value of this density is calculated as the sum of the values of each of the POIs or rovers divided by their euclidean distance from the sensor. The detections within each quadrant are used to compute the state input vector for the neural network controller. Formally, the state

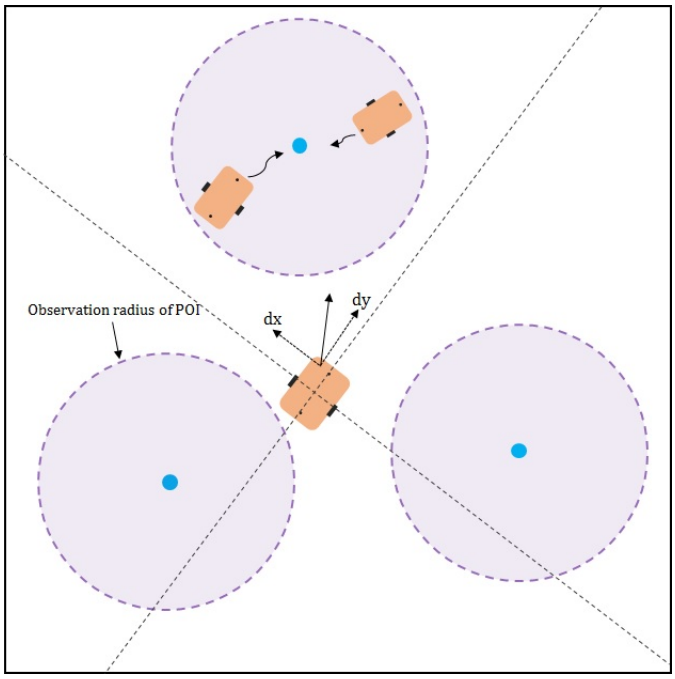


Figure 5.1: Diagram of the rover domain. The world is broken up into four quadrants relative to the rovers position and orientation. POIs and fellow rovers that are observed in each quadrant are summed resulting in 8 state input variables. At each time-step, the agent's neural network controller yields two continuous outputs $[d_x, d_y]$, which determine the rover's motion in the next time-step. Each POI has an observation radius such that only rovers within that radius are able to observe that POI.

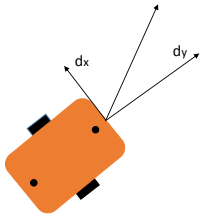


Figure 5.2: The figure shows the rover heading and the x and y axis with respect to the rover's heading. At each time step the rover has two continuous outputs (d_y, d_x) giving the magnitude of the motion in a two-dimensional plane relative to the rover's orientation.

variable representing rover detections in quadrant q for rover j is defined as:

$$s_{rover_{j,q}} = \sum_{j' \in J_q} \frac{1}{L_{j'}}, \tag{5.1}$$

where J_q is the set of the observed rovers in quadrant q, and $L_{j'}$ is the relative distance from rover j to rover $j^{'}$. The state variable representing POI detections in quadrant q of rover j is defined as:

$$s_{POI_{j,q}} = \sum_{i \in I_q} \frac{V_i}{L_i}, \tag{5.2}$$

where I_q is the set of observed POIs in quadrant q, and L_i is the relative distance between rover j and POI i. These state variables give an approximate representation of the world, reducing the location and number of rovers and POIs in each quadrant to a representative value.

5.2 Agent Action Representation

Every rover (agent) in this domain moves around in a continuous two-dimensional plane based upon its calculated actions. The actions are chosen by the agents' current policies, generated using the neuroevolutionary algorithm described in Section 2.3.3.1. The movement of each rover is governed by the neuroevolutionary algorithm as follows:

$$d_x = d(O_1 - 0.5), \tag{5.3}$$

$$d_y = d(O_2 - 0.5), \tag{5.4}$$

where $\frac{d}{2}$ is the maximum distance a rover can move in a given direction during a single time step, O_1 and O_2 are the x and y outputs from the agent's current neural network policy, and d_x and d_y are the action selections of the agent.

5.3 Objective Function

As discussed earlier, the main difference of this domain with previously explored multiagent rover domains in [2,32] is in the objective function, which requires tighter coordination among agents. Here, each POI requires multiple simultaneous observations within its observation radius to count as a success otherwise no reward is received by the team for those observations. In this domain, prior knowledge of POIs, such as their locations, observation radius, the number of POIs and the utility of observing any particular POI is not known and rovers must learn to coordinate such that the team's utility function is maximized.

To formalize the problem, we first focus on the simple case where $m = 2$ observations are required in order to observe a POI. In this case, if more than two rovers observe a POI, only the observations of the closest two rovers are considered and their observation distances are averaged in the computation of the global system evaluation G, which is formulated as:

$$G(\mathbf{z}) = \sum_i \sum_j \sum_k \frac{V_i N^1_{i,j} N^2_{i,k}}{\frac{1}{2}(\delta_{i,j} + \delta_{i,k})}, \tag{5.5}$$

where $\mathbf{z}$ is the joint state-action of the team, V_i is the value of observing the i-th POI, and $\delta_{i,j}$ is the distance between the j-th rover, and the i-th POI. The variables $N^1_{i,j}$ and $N^2_{i,k}$ indicate whether rovers j, k were within the observation distance δ_0 and were the closest two rovers to the i-th POI. That is,

$$N^1_{i,j} = \begin{cases} 1, & \text{if } \delta_{i,j} < \delta_0 \text{ and } \delta_{i,j} < \delta_{i,l},\ \forall l \neq j, \\ 0, & \text{otherwise,} \end{cases} \tag{5.6}$$

$$N^2_{i,k} = \begin{cases} 1, & \text{if } \delta_{i,k} < \delta_0 \text{ and } \delta_{i,k} < \delta_{i,l},\ \forall l \neq j,k, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

The overall team objective is to maximize the global system evaluation (5.5).

Each member of the team of rovers executes a policy described by a neural network. Neural networks have the ability to model continuous state-action control policies given only a coarse representation of the system state [4]. To train a neural network, the weights will be adjusted in such a way that given the current state as an input, the network returns an action that maximizes a particular utility function. The performance of the learned policy is strongly dependent on the utility function used to evaluate the policies.

In this work, we train neural network controllers with cooperative coevolutionary algorithms (CCEA) while we use the D_{++} evaluation function as the fitness function, and we compare the results against those trained on the difference (D) and global (G) evaluation functions.

5.4 Rover Policies

Neuro-evolutionary algorithms have been shown to be effective in domains with continuous states and actions [4] like the Cooperatively Coupled Rover Domain. In this work, agents policies are represented by neural networks and these policies are updated via CCEA algorithm explained in algorithm 2. Each agent evolves its policy from a pool of policies including 15 neural networks that are randomly initialized. Each of these neural networks has 9 hidden units, 8 inputs, 2 outputs, and each neuron utilizes sigmoid activation functions. Algorithm 2 shows the process that is used to select, mutate, evaluate the rover policies. Starting with random policies, in each generation, 10% of the neural network weights (randomly chosen) are mutated using a normal distribution of mean 0 and standard deviation of 1.0. Mutation results in shifting the population to cover more of the policy search space. Teams are created by selecting (without replacement) a policy from each agent's population, and are executed in the world for 20 time-steps. Depending on how well they performed the task, these policies are ranked according to the chosen policy evaluation function and are retained for the next generation using an ϵ-greedy policy selection.

5.5 Experimental Setup

To investigate the performance of the proposed shaped reward, different sets of experiments will be presented. We compare the performance of D_{++} learners against those using the difference evaluation function signal and global evaluation signal. The experiments are performed in simulation and the rover policies are evaluated and learned offline. Generally, at every time-step, each rover uses the policy encoded by the neural network controller to determine two controls in the (x, y) directions. The weights of the neural networks are adjusted via the (CCEA) algorithm, explained in algorithm 2. The eight inputs are the state variables as defined in (5.1) and (5.2). The neural network output provides two values in the range of $[0, 1]$ and each of these values are mapped to one of the two controls. Each agent has a population of 15 policies, initialized using a normal distribution $\mathcal{N}(0, 1)$.

In the following experiments, we first focus on the homogeneous multi-rover system for which the survey region is 30×30 units in size, containing a fixed number of randomly distributed POIs. Each POI requires a fixed number of simultaneous observations within its observation range, which is defined as $r_{POI} = 4.0$. The maximum allowable rover movement is defined as $d_{max} = 1$ unit/timestep. In this problem, we assume full observability for the rover sensors. In future work, we will address partial observability and its effect on the performance of the proposed learning algorithm.

5.6 Homogeneous Teams

Two sets of experiments are conducted. In both experiments, a team of 12 rovers explore the region to observe 10 POIs. In the first experiment, each POI requires 3 simultaneous observations. In the second experiment, in order to analyze how the learning algorithm performs with tighter agent coupling, the number of required observations for each POI was doubled.

5.6.1 12 Agents, 10 POIs, 3 Required Observations

Averaged learning results associated with all three reward functions, G, D, D_{++} with error bars reporting the standard error in the mean are given in figure 5.3. These results show that agents learning with the D_{++} evaluation signal significantly outperform

agents using global evaluations or the difference evaluation function alone. In fact, D_{++} produces policies that perform up to 100% better than those of D and at a quadruple speedup in the learning rate.

Figure 5.4 shows the paths of the team of rovers after being trained with the D_{++} reward function. The dots in magenta are the POIs, and they vary in size based on their value to the system. The POIs are spread across the exploration region. Rovers are initially located in the center and should spread and explore the area of interest. As seen in figure 5.4, rovers have successfully formed teams of three and have coordinated to observe multiple POIs in their proximity. The significance of these results is that team formation has been performed in an implicit manner, requiring no communication among agents or any prior information about other agents' intended behavior. This approach is particularly useful in cases where communication is limited or expensive since it can provide more reliable and robust results.

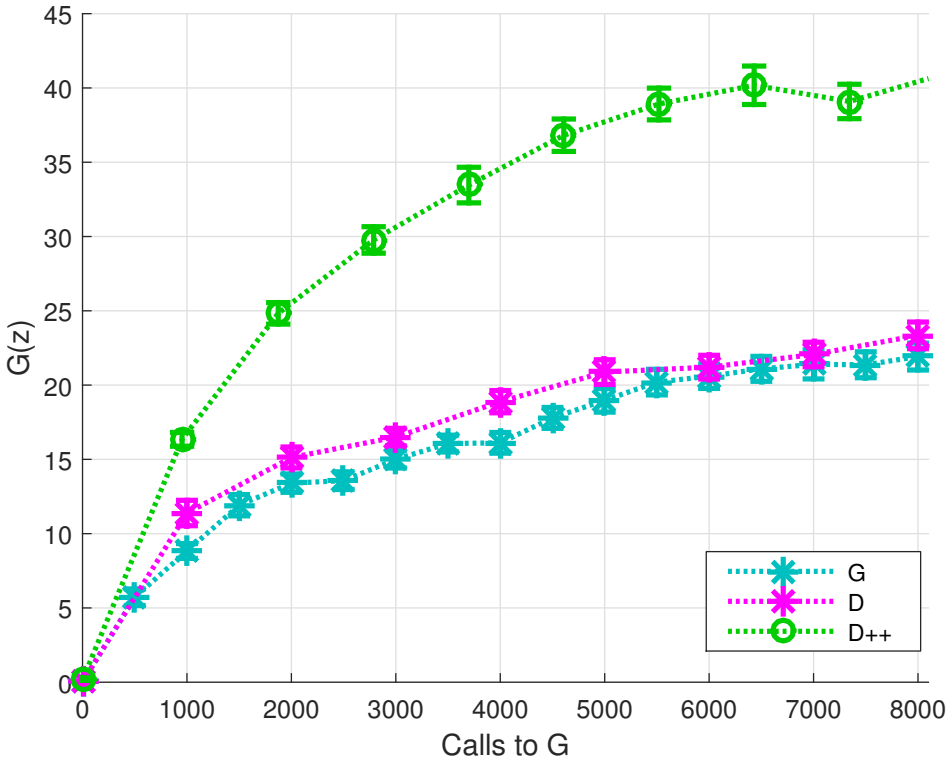


Figure 5.3: Observation performance of policies trained on G, D, D_{++} for 12 rovers, 10 POIs each requiring 3 simultaneous observations.

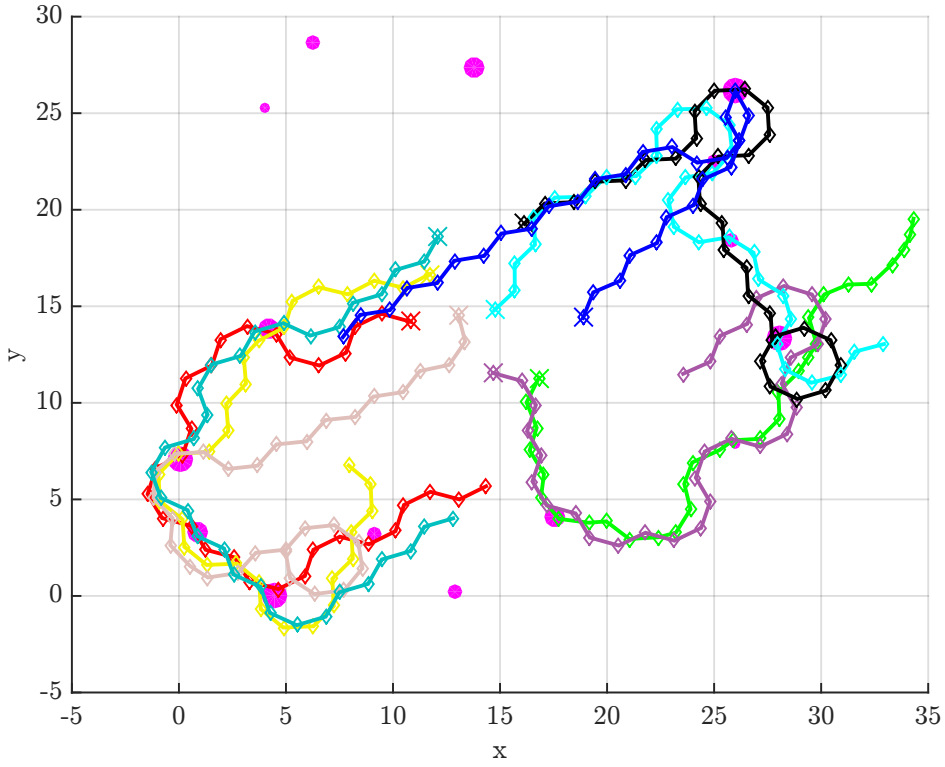


Figure 5.4: Rover paths executed by policies learned using the D_{++} reward function. POIs are represented as pink circles; larger circles indicate that the POI has a higher observation value.

5.6.2 12 Agents, 10 POIs, 6 Required Observations

To investigate the ability of the proposed algorithm in dealing with tighter agent couplings, we doubled the number of required simultaneous observations. Averaged results over 50 trials are shown in figure 5.5. These results indicate that as the coupling of the system increases, both G and D lose applicability since learning algorithms using either of these reward functions rely heavily on multiple agents simultaneously selecting the right actions, which is highly unlikely in tightly coupled domains. However, D_{++} overcomes this issue by using *counterfactual* agents. In particular in the rover domain, D_{++} rewards a policy that leads an agent to the observable region of a POI. Even if an insufficient number of rovers are available to accomplish the observation task; it promotes coordination by rewarding the *stepping stones* that ultimately lead to achieving the system objective.

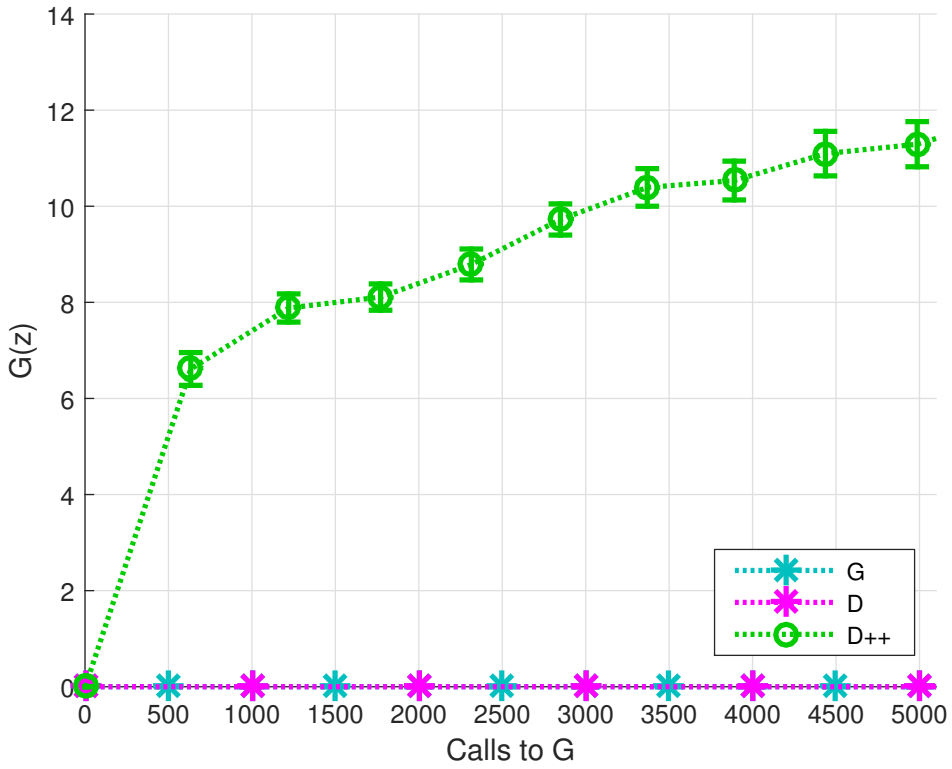


Figure 5.5: Observation performance of policies trained on G, D, D_{++} for 12 rovers, 10 POIs each requiring 6 simultaneous observations.

5.7 Computation Time Analysis

Recall from Section 3.4 that while calculating D_{++} one must continue introducing *counterfactual* agents until the resulting reward provides a gradient that can be used as the reward. However, as previous results on the performance of policies trained using D_{++} show, agents learn to coordinate relatively quickly compared to the results from using either G or D. This results in a decrease in the number of required *counterfactual* agents and also a decrease in the computation effort needed. Figure 5.6 supports the above argument by indicating the average required number of simulated *counterfactual* agents in D_{++} calculations during training. As shown in figure 5.6, there is a sharp decrease in the calculations required in D_{++} within the first 500 generations.

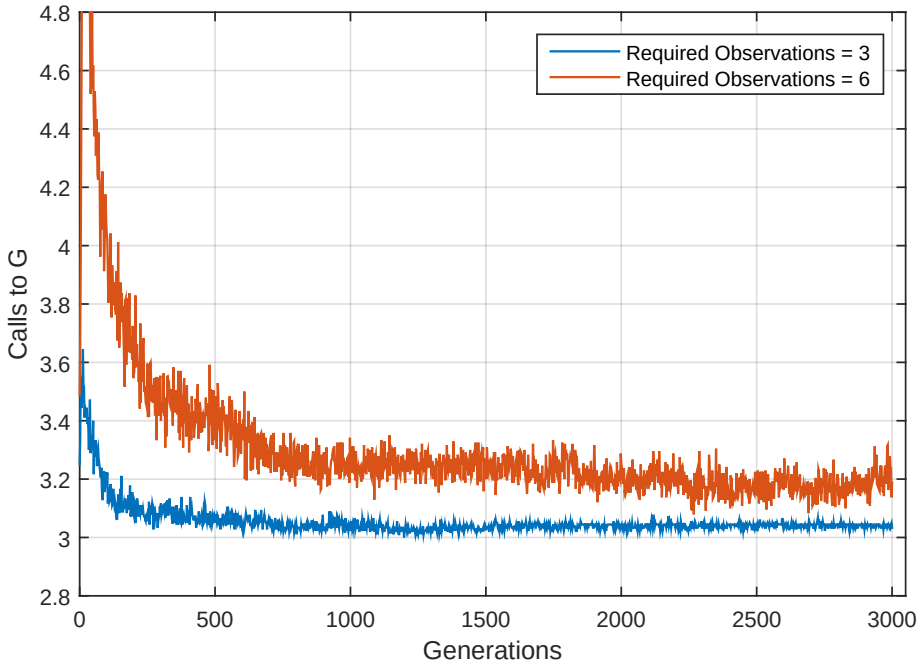


Figure 5.6: Average calls to G across progressive learning generations in the D_{++} calculation for two cases where POIs require 3 and 6 simultaneous observations, respectively. Averages over 50 trial runs are plotted along with the shaded region showing the 95% confidence interval.

5.8 Heterogeneous Teams

In this section, two sets of results on the heterogeneous exploration domain are presented. In both experiments, $t = 3$ types of rovers are considered. In the first experiment, each POI requires at least one rover of each type for a successful observation. In the second experiment, we increase the level of heterogeneity by requiring one rover of the first two types, $t = \{1, 2\}$, and three rovers of the third type, $t = 3$, to simultaneously observe a POI. This will indicate the power of our algorithm in dealing with different levels of system heterogeneity. Similar to the homogeneous case, the survey region is 30×30 units in size and each POI has a fixed observation radius of $r_{POI} = 4.0$. Again, the maximum allowable rover movement is $d_{max} = 1$ unit/time-step as defined before. The following results are generated from 50 statistical trials.

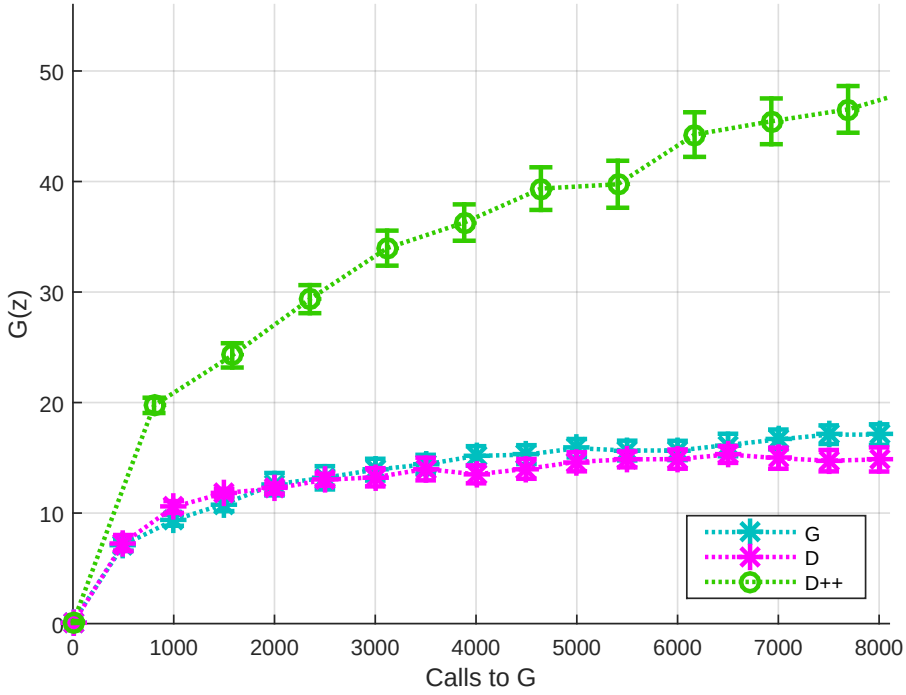


Figure 5.7: Observation performance of policies trained on G, D, D_{++} for 9 rovers, 15 POIs each requiring simultaneous observations from the rovers, one of each type t.

5.8.1 9 Agents, 15 POIs, 3 Required Observations of 3 Different Types

In this experiment, we assume that there are 3 different types of rovers, each having a particular capability required for the successful observation of the POIs. A POI is counted as observed only if it is simultaneously observed by at least one rover of each type. Figure 5.7 indicates the learning curves comparing the performance of policies learned by three different functions, G, D and D_{++}. The learning curves are plotted against number of calls to G to account for the required computation of each reward function. As seen, with the same number of calls to G, policies learned using G and D perform similarly while D_{++} yields policies that perform about 1.5 times better and up to four times faster compared to the results of G and D.

Figure 5.8 is a snapshot of the exploration domain which depicts the actual rover

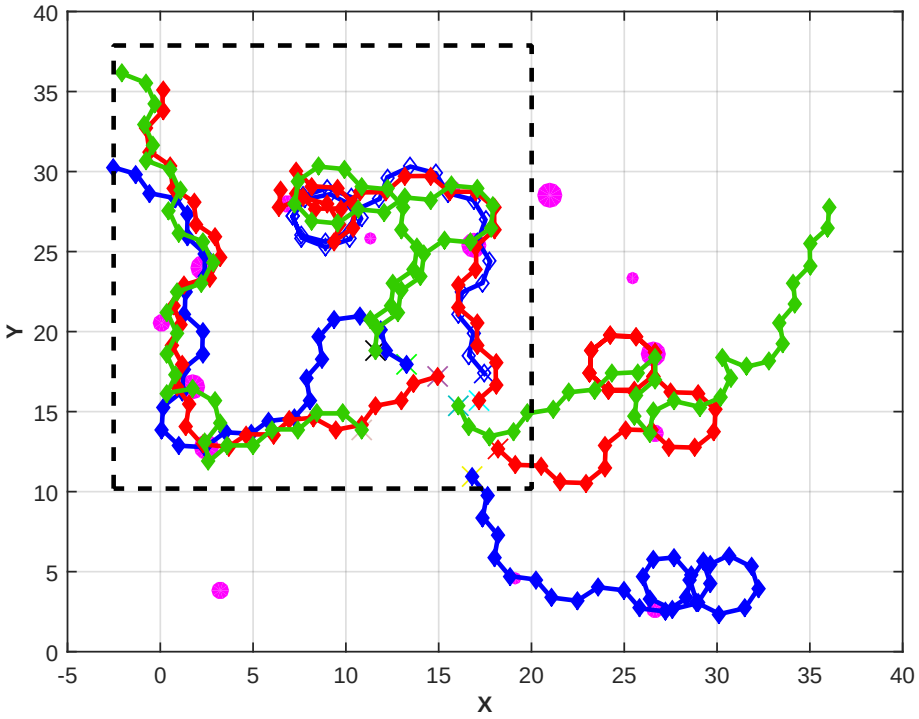


Figure 5.8: Rover paths executed by policies learned using the D_{++} reward function. Different colors of *blue*, *green* and *red* are used to represent different types of rovers in the system. As seen, two groups of rovers, bounded in the dashed box, consist of all the three required types and have successfully formed teams to explore the POIs in that region. However, the remaining rovers are still optimizing their policies trying to observe the remaining POIs.

paths executed by policies learned using the D_{++} reward function. The path are colored in *blue*, *green* and *red*, each representing a different type of rover in the system. The POIs are marked with magenta dots with varying sizes, indicating their importance to the system. As seen, the rovers, starting from the center of the domain, have formed two distinctive teams consisting of one rover of each type which have successfully coordinated to make observations of the POIs along their paths. Meanwhile, three rovers, shown outside of the dashed box, have not fully coordinated.

5.8.2 9 Agents, 15 POIs, 5 Required Observations of 3 Different Types as ([1, 1, 3])

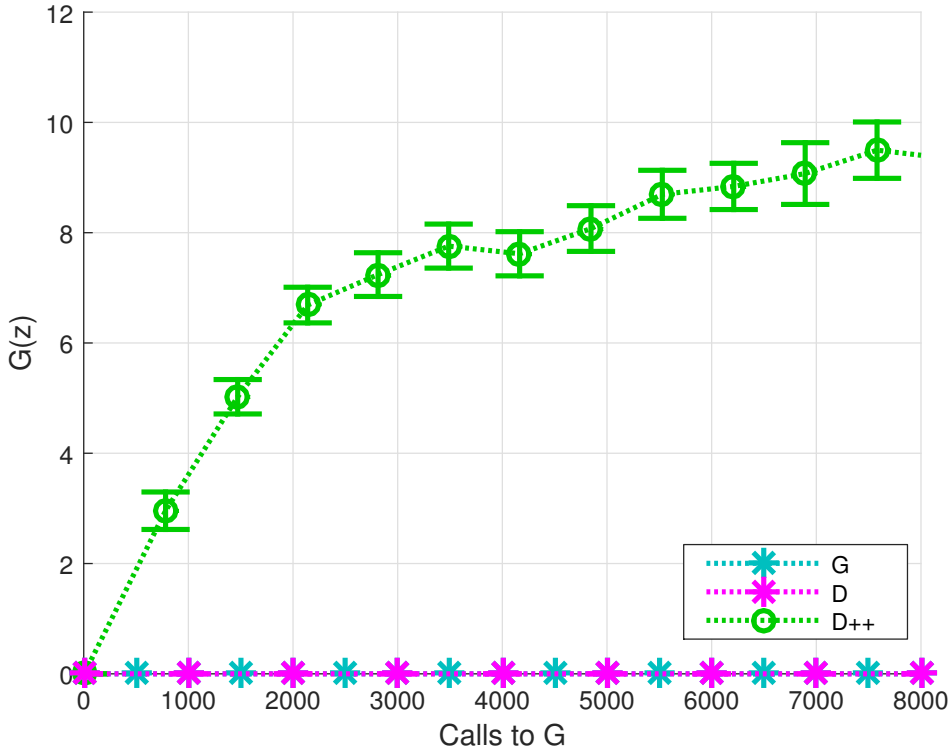


Figure 5.9: Observation performance of policies trained on G, D, D_{++} for 9 rovers, 15 POIs each requiring simultaneous observations of a total of 5 rovers, including one of each types of $t = 1$ and $t = 2$, and three rovers of type $t = 3$.

This experiment measures the performance of the D_{++} algorithm when the level of the system heterogeneity increases. For this purpose, we assume that each POI must be simultaneously observed by at least five distinctive rovers, one from each of the first two types, and three of the third type. An equivalent problem in a real world scenario can be where we have different rover types each equipped with different sensors and tools to explore an unknown area. In such a case, three rovers can assist in localization, and the second and third type may provide the tools to perform excavation and data collection. Figure 5.9, indicates the performance of the learned policies in stimulated scenario. It is clear from these results that training using either G or D fails to yield a high-reward policy. This can be explained by the fact that both of these reward functions rely on

multiple agents simultaneously selecting the right actions, which is highly unlikely in such a tightly coupled problem. The problem is even more severe compared to the homogeneous domain since the rover types also matter and any coordination algorithm must account for that as well. In contrast to G and D, D_{++} manages to reward policies that are *stepping stones* to the ultimate system objective even in the case of agent heterogeneity.

5.9 Robustness Analysis

In this section, we investigate how failure of agents impacts learning. To this end, after 2000 generations, we fix 25% of the agents (the agents are static in their initial position and do not move). In this experiment, there are 12 agents, 10 POIs scattered across the domain, each requiring 3 simultaneous observations. The results are averaged over 50 statistical runs.

It is noteworthy, that failure percentage has been intentionally set high to greatly challenge coordination afterwards. Since it is tightly coupled task, the failure of agents greatly impairs the system. Figure 5.10 indicates the results of the robustness analysis which shows that although D_{++} outperform the other two reward functions, they seemingly suffer greater than G or D learners in the case of failure. This can be explained by the fact that those failed agents have actually coordinated and are achieving utility for the system and as such, the failure of those agents greatly impacts the systems performance. This demonstrates that D_{++} makes good use of all of the agents in the team. On the other hand, failed agents that were trained using the G and D learners are mostly randomly exploring agents that have not coordinated well with other team members, and their removal does not severely impact team performance. In terms of recovering from the failure, it can be observed that D_{++} learners recover less compared to G and D learners. This behavior can be explained with the tightly coupled nature of the task. It is obvious that by failure of 25% agents in such a tightly coupled system, the maximum achievable performance drops and even a perfect coordination algorithm is not able to yield to the same performance as before. However, in the case of G and D learners, since the resources are not fully utilized even after 2000 generations, there is still room for improvement in behavior.

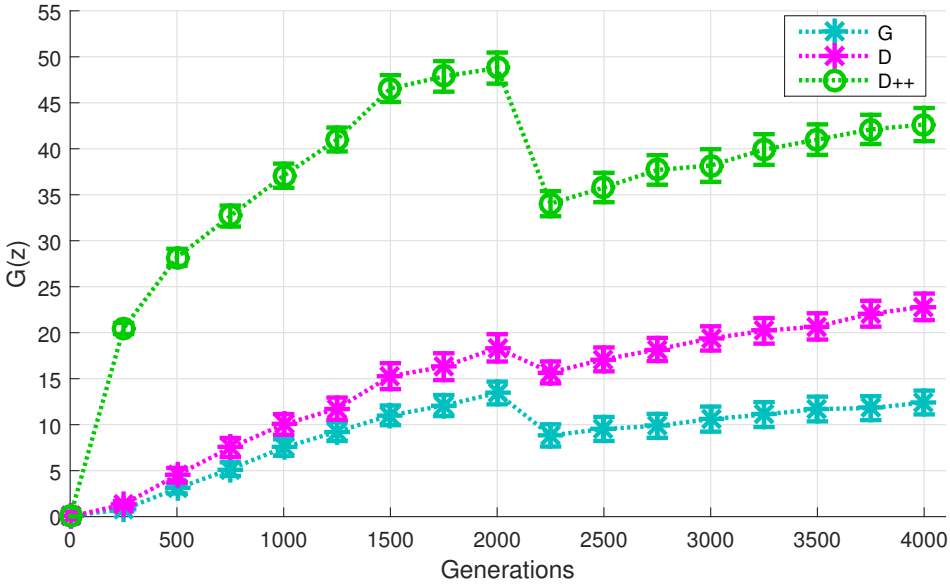


Figure 5.10: Observation performance of policies trained on G, D, D_{++} for 12 agents, 10 POIs each requiring simultaneous observations of a total of 3 agents. At 2000 generations, 25% of agents fail (stay static).

Chapter 6: Conclusion

6.1 Summary

Coordinating disparate agents such that can collectively achieve a complex task is a key problem in multiagent settings that can open up more advanced usage of multiagent systems in many real-world applications. Even though there are many mechanisms developed mainly for solving such coordination problems, they are often unable to fully address the coordination issues due to limiting communication restrictions, insufficient domain knowledge or the general problem complexity that make it increasingly more difficult to define good agent policies a priori.

We identified one of the major setbacks in tightly coupled agent coordination which is the fact that the chances of agents finding the proper joint actions required to achieve the task is very low and as such, agents do not receive strong feedback throughout learning. Therefore, in order to improve the coordination task we focused on the structural credit assignment problem in tightly coupled multiagent domains and we built upon current work in the literature of multiagent credit assignment to develop a reward structure that enables us to reward stepping-stone policies of agents regardless of the task being fully accomplished or not.

D_{++} is a novel reward framework which builds on the idea of *counterfactuals* and aims to differentiate between actions that are potentially useful in the long term goal achievement and actions that are not beneficial to the system. To indicate the ability of the proposed reward function in training good and coordinated agent policies, we demonstrated the performance of the algorithm in two different problem domains defined based on a tightly coupled environmental monitoring task.

In the first problem domain, which was a simplified version of the continuous cooperative rover domain, we mainly explored the impact of different objective functions and we indicated the how the degree of coupling impacts learning for different objectives. We also explored the performance of the D_{++} algorithm in the continuous rover domain which poses larger coordination challenges and we investigated the problem along two

dimensions, the heterogeneity of the multiagent system and the degree of coupling. D_{++} was successfully tested in the homogeneous domain, showing significant improvement in learning, both in terms of convergence speed and learning performance, compared to the policies trained on either G and D. Additionally, to account for system heterogeneity, the D_{++} reward function was extended to a domain with different types of agents, all required for the completion of the task. D_{++} was also able to successfully learn coordinated policies in this domain whereas both G and D fail to find high-reward policies.

The results presented in this work, confirmed our original hypothesis that by rewarding the *stepping stones* policies one can highly promote coordination. Notably, as the coupling of the system increases, both G and D lose applicability since they rely heavily on multiple agents simultaneously selecting the appropriate actions, which is highly unlikely in tightly coupled domains.

Finally, we investigated how failure of agents impacts learning. The results of the robustness analysis indicated that D_{++} learners suffer greater than G or D learners which was explained by the fact that those failed agents had actually coordinated and were achieving utility for the system. Such greater failure impact indicates good use of the resources whereas in the cases of G and D learners, the failed agents are mostly randomly exploring agents that have not coordinated well with other team members and their removal does not impact the performance as much.

6.2 Future Work

In this work, we showed how the use of *counterfactual* agents provides a powerful tool to tackle different multiagent coordination problems. Although most of the present work focuses on homogeneous teams of agents, we showed that by extending the *counterfactual* search domain, D_{++} algorithm can also account for agent heterogeneity. These results are encouraging to further extend this research to a broader range of problems, for example, cases with heterogeneous actions. An example of this case in the Cooperatively Coupled Rover Domain is where agents are necessarily required to observe a POI from different angles. By adding heterogeneity to the agents' actions, D_{++} computation can rise quickly. A possible solution to this problem is to come up with better *counterfactuals*. For example, in the present work, the *counterfactual* was defined as agents that are located in the same position as the primary agent and executing the same policy. For

another domain, and depending on the agents' states and actions, other definitions of *counterfactuals* may be required. Also, in this work, we showed that we can greatly reduce the computation by making a simple assumption that at any state the entire team of agents must be able to accomplish the task if at all possible. This is a valid assumption if the problem is well-posed. Similar assumptions can be made to guide the search when calculating D_{++} to decrease the computation for other types of domains. In addition, as discussed in section 3.4, D_{++} calculations quickly drop as learning proceeds and the reason is that by proper rewarding, it is able to coordinate the agents comparatively quickly which saves a great deal of computation afterwards.

Furthermore, throughout this work, we assumed full observability for the entire team. In [9], authors showed that D can be approximated from locally available information. Future work will investigate how partial observability impacts learning in tightly coupled tasks and how D_{++} can be approximated from local information.

Bibliography

[1] Noa Agmon, Sarit Kraus, and Gal A Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2339–2345. IEEE, 2008.

[2] Adrian Agogino and Kagan Tumer. Efficient evaluation functions for multi-rover systems. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 1–11. Springer, 2004.

[3] Adrian Agogino and Kagan Tumer. Multi-agent reward analysis for learning in noisy domains. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 81–88. ACM, 2005.

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

[5] Adrian K Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[6] Michael Bowling and Manuela Veloso. Simultaneous adversarial multi-robot learning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, volume 3, pages 699–704, 2003.

[7] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[8] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, (s 746):752, 1998.

[9] Mitchell Colby, Jen Jen Chung, and Kagan Tumer. Implicit adaptive multi-robot coordination in dynamic environments. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5168–5173. IEEE, 2015.

[10] Mitchell Colby and Kagan Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 425–432. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[11] Mitchell Colby and Kagan Tumer. Fitness function shaping in multiagent cooperative coevolutionary algorithms. *Autonomous Agents and Multi-Agent Systems*, pages 1–28, 2015.

[12] Mitchell K Colby, Ehsan M Nasroullahi, and Kagan Tumer. Optimizing ballast design of wave energy converters using evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1739–1746. ACM, 2011.

[13] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 225–232. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[14] Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 433–440. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[15] Shane Farritor and Steven Dubowsky. A genetic algorithm-based navigation and planning methodology for planetary robotic exploration.

[16] Sevan G Ficici, Ofer Melnik, and Jordan B Pollack. A game-theoretic and dynamical-systems analysis of selection methods in coevolution. *Evolutionary Computation, IEEE Transactions on*, 9(6):580–602, 2005.

[17] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

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

[19] Brian P Gerkey and Maja J Matarić. Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 464–469. IEEE, 2002.

[20] Brian P Gerkey and Maja J Mataric. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3862–3868. IEEE, 2003.

[21] Dani Goldberg and Maja J Matarić. Maximizing reward in a non-stationary mobile robot environment. *Autonomous Agents and Multi-Agent Systems*, 6(3):287–316, 2003.

[22] Uli Grasemann, Daniel Stronger, and Peter Stone. A neural network-based approach to robot motion control. In *Robot Soccer World Cup*, pages 480–487. Springer, 2007.

[23] Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2004.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[25] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[26] Matt Knudson and Kagan Tumer. Coevolution of heterogeneous multi-robot teams. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 127–134. ACM, 2010.

[27] Evelina Lamma, Fabrizio Riguzzi, and L Pereira. Belief revision by multi-agent genetic search. In *In Proc. of the 2nd International Workshop on Computational Logic for Multi-Agent Systems, Paphos, Cyprus*, 2001.

[28] Alan Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical report, 1987.

[29] Maja J Mataric. Reward functions for accelerated learning.

[30] J-B Mouret and Stéphane Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

[31] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, pages 278–287, 1999.

[32] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[33] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240, 1998.

[34] Lynne E Parker. Lifelong adaptation in heterogeneous multi-robot teams: Response to continual variation in individual robot performance. *Autonomous Robots*, 8(3):239–267, 2000.

[35] David C Parkes and Satinder Singh. An mdp-based approach to online mechanism design. 2004.

[36] J Pieter and Edwin D de Jong. Evolutionary multi-agent systems. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 872–881. Springer, 2004.

[37] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel problem solving from naturePPSN III*, pages 249–257. Springer, 1994.

[38] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[39] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

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

[41] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[42] Kagan Tumer and Adrian Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 591–598. ACM, 2005.

[43] Karl Tuyls, Pieter Jan 'T Hoen, and Bram Vanschoenwinkel. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Autonomous Agents and Multi-Agent Systems*, 12(1):115–153, 2006.

[44] Jijun Wang, Michael Lewis, and Paul Scerri. Cooperating robots for search and rescue. In *Agent Technology for Disaster Management Workshop at AAMAS*, volume 6, 2006.

[45] L Darrell Whitley, Frederic Gruau, and Larry D Pyeatt. Cellular encoding applied to neurocontrol.

[46] David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(02n03):265–279, 2001.

[47] David H Wolpert, Kagan Tumer, and Jeremy Frank. Using collective intelligence to route internet traffic. *arXiv preprint cs/9905004*, 1999.

[48] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.