AN ABSTRACT OF THE DISSERTATION OF

Chris HolmesParker for the degree of Doctor of Philosophy in
Mechanical Engineering presented on April 15, 2013.

Title:

CLEAN Learning to Improve Coordination and Scalability in Multiagent Systems

Abstract approved: _____

Kagan Tumer

Recent advances in multiagent learning have led to exciting new capabilities spanning fields as diverse as planetary exploration, air traffic control, military reconnaissance, and airport security. Such algorithms provide a tangible benefit over traditional control algorithms in that they allow fast responses, adapt to dynamic environments, and generally scale well. Unfortunately, because many existing multiagent learning methods are extensions of single agent approaches, they are inhibited by three key issues: i) they treat the actions of other agents as "environmental noise" in an attempt to simplify the problem complexity, ii) they are slow to converge in large systems as the joint action space grows exponentially in the number of agents, and iii) they frequently rely upon the presence of an accurate system model being readily available.

This work addresses these three issues sequentially. First, we improve overall learning performance compared to existing state-of-the-art techniques in the field

by embracing the exploration in learning rather than ignoring it or approximating it away. Within multiagent systems, exploration by individual agents *significantly alters* the dynamics of the environment in which all agents learn. To address this, we introduce the concept of "private" exploration, which enables each agent to present a stationary baseline policy to other agents in order to allow other agents in the system to learn more efficiently. In particular, we introduce Coordinated Learning without Exploratory Action Noise (CLEAN) rewards which improve coordination and performance by utilizing the concept of private exploration in order to remove the negative impact of traditional "public" exploration strategies from learning in multiagent systems. Next, we leverage the fundamental properties of CLEAN rewards that enable private exploration to allow agents to explore multiple potential actions concurrently in a "batch mode" in order to significantly improve learning speed over the state-of-the-art. Finally, we improve the real-world applicability of the proposed techniques by reducing their requirements. Specifically, the CLEAN rewards developed require an accurate partial model (i.e., an accurate model of the system objective) of the system in order to be computed. Unfortunately, many real-world systems are too complex to be modeled or are not known in advance, so an accurate system model is not available a priori. We address this shortcoming by employing model-based reinforcement learning techniques to enable agents to construct their own approximate model of the system objective based upon their observations and use this approximate model to calculate their CLEAN rewards.

CLEAN Learning to Improve Coordination and Scalability in Multiagent Systems

by

Chris HolmesParker

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 15, 2013
Commencement June 2013

Doctor of Philosophy dissertation of Chris HolmesParker presented on April 15, 2013.

APPROVED:

Major Professor, representing Mechanical Engineering

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

---

Dean of the Graduate School

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

Chris HolmesParker, Author

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Kagan Tumer for his very valuable insight on multiagent techniques, for his support, and for always pushing me to improve the quality of my work.

Additionally, I would like to thank my committee, Dr. Belinda Batten, Dr. Ross Hatton, Dr. Jonathan Hurst, Dr. Ravi Balasubramanian, and Dr. Richard Nafshun for their time and patience throughout this project.

I would like to thank the AADI lab for their help and very useful comments.

I would like to extend a special thank you to the Portland chapter of the Achievement Rewards for College Scientists (ARCS) foundation, without whom my time in graduate school would not have been possible.

I would like to thank a number of people who have advised me over the years, including: William Reiersgaard, Tom Overman, Robert Cava, Douglas Keszler, Rich Josephson, William Hetherington, Brian Bloudek, Frank Oliver, Nahum Gat, Linda Papermaster, Joel Gat, Billy Bloudek, Adrian Agogino, and Orville Holmes. The support, advice, and help I have received from these people has been extremely valuable and is the backbone behind my success.

I would like to thank my parents (Luana Parker, Martin Parker, and Larry Holmes) and siblings (Lisa, Jessica, Jerry, and Orville) for all their support throughout the years.

And last but not least, I would like to thank my amazing girlfriend Ilana Gat for all her support, for her encouragements, for being there for me, and for being my best friend.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Chapter 1 – Introduction

Multiagent learning algorithms have gained popularity and acceptance in a variety of commercial (e.g., air traffic management), industrial (e.g., the "smart" electrical grid), military (e.g., distributed sensing), and scientific (e.g., fractionated satellites) domains. The complexity of tasks within these systems render preplanned agent control techniques inadequate as they are either too computationally expensive to compute or simply too slow to respond to the rapidly changing environmental dynamics. Instead, agents must discover their own control solutions, using learning approaches, where they continually interact with their environment in order to learn a mapping from their states to actions. As most multiagent learning algorithms are extensions of single agent learning algorithms, they consist of agents taking actions and updating their internal parameters based on the reward they receive. The agents continually strike a balance between *exploiting* the knowledge they have already gained (i.e., taking actions that currently they believe will lead to good performance) and *exploring* new actions which may improve their knowledge of the system and eventually lead to better performance. It is known that learning agents must balance these two behaviors and this has become known as the "exploration-exploitation" tradeoff in learning.

In traditional single-agent reinforcement learning, an agent consistently takes exploratory actions to ensure that it does not prematurely conclude that a sub-

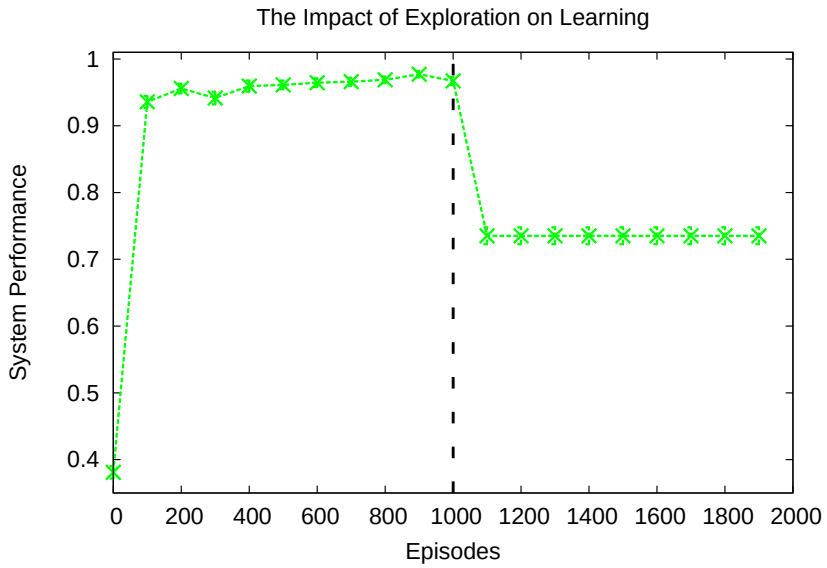

Figure 1.1: When an agent "exploits" its policy, it takes its best known action. When an agent "explores," it takes actions that it is less sure of in hopes that it will discover a good action. In single agent learning, once learning is complete, using the exploitive policy will almost always lead to higher performance since it is the best known policy. This is often not the case in multiagent learning since an agent will be exploiting a policy that assumed all other agents were exploring. In this figure, even on a relatively simple problem, performance actual goes down at the end of learning (episode 1000) when exploration is turned off and all agents exploit their best policy.

optimal solution is its best option. Over time, these exploratory actions help the agent learn the real underlying values associated with its actions. In fact, it has been shown that in single agent learning, such exploration is necessary to converge to an optimal policy. Unfortunately, in multiagent systems, the exploratory actions of individual agents fundamentally alter the dynamics of the system the agents are attempting to learn (see Figure 1.1). Indeed, if an agent treats the other agents in the system as part of the environment, it has no way to distinguish between

environmental noise and the noise associated with exploratory actions of other agents. Learning what other agents are likely to do in a given state is critical for the success of a learning agent. But learning the exploratory noise of the other agents' actions is not only difficult, but also self-defeating. In such a case, the agents are attempting to learn an artifact, one that will disappear when the agents start to converge to their true policies.

In this work, we leverage our previous work on credit assignment to develop Coordinated Learning without Exploratory Action Noise (CLEAN) rewards, which address the exploratory noise problem [59, 7, 60, 63]. These rewards are based on the concept of counterfactual actions that to date have been used to enable individual agents to determine their individual impact on overall system performance. CLEAN rewards extend and generalize this concept in order to allow agents to have private exploration, where they explore the values of their own actions without impacting the learning of other agents. Overall, this algorithm lets agents distinguish when their peers are taking purposeful actions (e.g., exploiting their current knowledge) from when they are taking exploratory actions (e.g., randomly sampling the action space to attempt to improve their system knowledge) by utilizing counterfactuals to privatize exploration, and as we show, provides a tremendous leap in multiagent performance.

A second, related problem in multiagent learning is the slow convergence of the system to good joint actions. This is partially caused by the noise associated with the exploration/exploitation trade-off described above, but also by the sheer size of the joint state-action space. Indeed, that space grows exponentially with the

number of agents, and a slow, traditional search through the solution space proves difficult, if not impossible, when the number of agents grow into thousands. In this work, we extend the concept of private exploration detailed above to enable agents using CLEAN rewards to update their perceived valuations of multiple actions at a time, resulting in "batch" updates. This is done by exploiting the concept of "privatized exploration" via counterfactuals introduced with CLEAN rewards. In particular, this enables agents to utilize counterfactuals combined with a reward model and system state information to approximate the impact they would have had on the system had they taken one of a set of alternate actions, resulting in a rewards for "potential" actions that were not explicitly taken. Allowing agents to perform multiple concurrent updates enables them to search the joint state-action space much more quickly and efficiently, which we show significantly improves learning speed.

Finally, although the CLEAN rewards described above result in significantly improved performance and learning speed, they have a drawback in that they require an accurate partial system model (i.e., an accurate model of the system objective) in order to be computed. In fact, the requirement of either an accurate full or partial system model is a key drawback of many existing multiagent learning techniques, and is a factor which significantly inhibits the real-world applicability of many multiagent techniques. For our final contribution, we extend the applicability of CLEAN rewards into real-world domains by enabling agents to learn without being given an accurate system model a priori. Here, we enable agents to utilize statistical function approximation tools to construct their own ap-

proximate model of the system objective through repeatedly interacting with their environment. Agents are then able to use their approximate models of the system objective to calculate their CLEAN rewards. In this setting, agents utilize the approximate model of the system objective they create to calculate their individual CLEAN rewards. As we show, the performance of agents using CLEAN rewards with an approximate model of the system objective is comparable in performance to existing state-of-the-art reward shaping techniques that are given an accurate system model.

1.1 Contributions

Overall, these three issues have inhibited the long-term goal of the field of multiagent systems, which is to enable decentralized control over systems comprised of thousands of disparate agents. In this work, we directly address all three of these issues. First, we formulate a paradigm where we shift from a "public" exploration strategy (i.e., all agents explicitly take exploratory actions and all agents observe each other's exploratory actions) to a "private" exploration strategy (i.e., agents do not explicitly take exploratory actions, instead, agents implicitly take exploratory actions which are not visible to other agents). In particular, we introduce Coordinated Learning without Exploratory Action Noise (CLEAN) rewards, which leverage the properties of a "private" exploration strategy in order to remove learning noise and improve coordination in multiagent systems. Second, we exploit the properties of CLEAN rewards in order to enable agents to obtain

"batch" rewards for multiple possible actions during each of their interactions with the environment, resulting in significantly improved learning speed. Finally, we use model-based reinforcement learning techniques to extend the applicability of these algorithms into real-world systems where accurate system models are not readily available and agents must learn within an unknown environment. This work is aimed at improving the capabilities of multiagent systems, as well as their real-world applicability. The particular contributions of this work are as follows:

- We identify, formalize, and define the *exploratory action noise* caused by agent exploration in multiagent systems which is a previously unidentified issue that has been plaguing the field of multiagent systems.
- We design a paradigm shift away from explicit "public" agent exploration strategies towards "private" implicit exploration strategies to address *exploratory action noise*.
- We introduce and define Coordinated Learning without Exploratory Action Noise (CLEAN) rewards which eliminate *exploratory action noise* in multiagent learning, increasing system performance over existing state-of-the-art techniques (e.g., difference rewards).
- We introduce Batch-CLEAN, which leverages the properties of "private" exploration to enable agents to perform multiple learning updates (i.e., control updates) per interaction with their environment, increasing learning speed over the state-of-the-art (e.g., difference rewards).

- We combine the CLEAN rewards paradigm with statistical function approximation techniques which enable agents to utilize these methods in unknown environments, improving the real-world applicability of these techniques over the existing state-of-the-art (e.g., potential-based reward shaping, difference rewards).

The remainder of this work is structured as follows. Chapter 2 provides background information on various aspects of multiagent systems and multiagent learning[1]. Chapter 3 introduces the issue of *exploratory action noise* and presents Coordinated Learning without Exploratory Action Noise (CLEAN) rewards as a solution to this problem. Chapter 4 demonstrates the performance of CLEAN rewards in two toy domains and demonstrates that CLEAN rewards outperform existing state-of-the-art techniques (e.g., difference rewards). Chapter 5 proposes an extension of the concepts CLEAN rewards that leverages the concept of private exploration to enable agents to perform "batch" reward updates to significantly improve learning speed. Then, Chapter 6 demonstrates the performance of these Batch-CLEAN rewards in two real-world domains, comparing their performance to both standard CLEAN rewards as well as difference rewards. Chapter 7 outlines a methodology that extends the applicability of CLEAN based techniques into real-world domains with unknown environments by implementing model-based reinforcement learning techniques to enable agents to construct their own environmental models based upon their own observations instead of relying upon an accurate system model

[1]This chapter can be skipped if the reader is already familiar with multiagent systems and multiagent learning

to be given a priori. Chapter 8 then demonstrates the performance of CLEAN rewards utilizing model-based reinforcement learning techniques to learn within initially unknown environments using two real-world domains. Finally, Chapter 9 provides a discussion of the conclusions of this work, along with opportunities for future work.

Chapter 2 – Background

2.1 Agent Learning

Throughout this dissertation, we are interested in intelligent agents which are able to autonomously "learn" their own control policies (i.e., agents that are able to learn how to behave and act within an environment). Within this section, we will be introducing one of the most common frameworks for these learning based problems (e.g., Markov Decision Processes) and then outlining some standard approaches to agent-based learning (e.g., reinforcement learning). This section will outline the fundamental properties of single-agent learning systems and will set the stage for our introduction to learning within multiagent environments.

2.1.1 Markov Decision Processes (MDPs)

A Markov Decision Process (MDP) is a framework that can be used to describe a learning problem for agents that are learning within a stochastic environment. The MDP framework was created when planning methods failed to handle stochasticity in the environment and actions. Instead of transitioning deterministically from one state to another given a certain action, when the actions in a given state lead to a series of potential future states s' for an action taken. An MDP is a four-tuple $\{S, A, R, T\}$, consisting of a set of agent states S, actions A, a deterministic

reward function R, and a transition probability function T. The set of states S, is the complete set of states that an individual agent can observe itself in. The set of actions A, is the complete set of actions each individual agent is capable of taking. The reward function R provides a direct mapping from the observed current state of an agent s to the reward r associated with being in that state. Finally, T contains the probability of ending up in state s' at time $t+1$ after taking action a in state s at time t. This four-tuple characterizes a Markov Decision Process. MDP's are capable of handling stochasticity and environmental uncertainty associated with state-transitions (i.e. action a in state s will not always yield a specific state s') [116].

2.1.2 Reinforcement Learning

Reinforcement learning can be considered a computational approach to understanding and automating goal-directed learning and decision making [125].[1] Reinforcement learners observe the state of the environment and attempt to take actions that maximize their expected future reward.[2] A reinforcement learning problem consists of at least one agent and an environment. A reinforcement learning agent has four key elements including a *policy*, *reward function*, *value function*, and optionally an *environmental model* [125].

An agent's *policy* defines the way the agent behaves at any given time [125],

[1] A comprehensive list of single and multiagent reinforcement learning algorithms can be found in [30, 125, 154].

[2] In this work, an agent's environment consists of the world and all other agents and the problem is represented as a four-tuple MDP.

it can be thought of as the agent's controller. This controller maps the agent's perceived environmental state to the action it takes in that state. Initially, these policies are initialized in an arbitrary manner, and adjusted over time as the agent learns. Each agent's policies are updated via the agent's *reward* and *value function*.

An agent's *reward* function reflects the agent's goal in a reinforcement learning problem [125]. The reward function provides an agent with a learning signal for actions taken. In general, the learning signal is positive if the agent's actions were beneficial to its goal, and negative if the actions were detrimental to the agent's goal. The rewards an agent receives are frequently coupled with a value function in order to update the agent's policy (controller).

A reinforcement learning agent uses some form of a *value function* in order to update its policy based upon the reward it receives. There are many forms of value updates in reinforcement learning, depending upon the domain. Here, we will introduce one of the most common forms of value functions known as a Q-function. At every episode an agent takes an action and then receives a reward evaluating that action. Agents select the actions corresponding to the highest Q-value with probability $1 - \epsilon$, and chooses a random action with probability ϵ. The constant ϵ is an exploration rate. After taking action a and receiving reward R an agent updates its Q table (which contains its estimate of the value for taking action a in state s [125]) via the Q-update function as follows:

$$Q'(s,a) = Q(s,a) + \alpha \left(R(s) - Q(s,a) + \gamma Q_{max}(s', a') \right) \tag{2.1}$$

where,

- $Q'(s, a)$ is the updated Q-value for taking action a in state s at time t
- $Q(s, a)$ is the current Q-value for taking action a in state s at time t
- $max_{a'}Q(s', a')$ is the maximum possible Q-value associated with taking action a' in state s' at time $t + 1$
- $R(s)$ is the reward received for the agent being in state s at time t
- α is the learning rate $\{0, 1\}$
- γ is the discount factor $\{0, 1\}$

Each of the $Q(s, a)$ values provides an agent with a measure of the amount of reward it can expect to achieve over time for taking action a in state s. The learning rate α controls how quickly an agent learns. If α is set low, new rewards will not impact the agents $Q(s, a)$ values as quickly, resulting in slower learning and slower changes in behavior. This is useful in domains where The environment changes slowly over time. When the α parameter is set high, learning occurs rapidly, pushing the $Q(s, a)$ values to change quickly with respect to rewards received. This is very useful in domains where the environment is changing rapidly, and the agents policy needs to change accordingly. The discount factor γ on the other hand impacts how far ahead an agent looks when considering its actions. A discount factor of $\gamma = 0$ will consider only the immediate reward obtainable from taking an action a in the current state s. A higher discount factor causes an agent to consider

the down-stream effects of its actions, how the action it takes in the current state s will impact the cumulative reward it receives in the future.

2.1.3 Model-Based Reinforcement Learning

Model-based reinforcement learning is of particular interest to this work. Model-based Reinforcement Learning refers to learning optimal behavior indirectly by learning a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward [107]. The models predict the outcomes of actions and are used in lieu of or in addition to interaction with the environment to learn optimal policies [107]. Many existing reward shaping techniques (e.g., Q-learning [125]) allow agents to learn without a model of the environment, simply through repeated interactions. Unfortunately, in many domains (e.g., multiagent reinforcement learning domains) this type of learning can be prohibitively slow, as each piece of information agents receive from the environment is used once and then discarded. To address this shortcoming, there has been a plethora of research in the area of model-based reinforcement learning [84, 102, 75, 107, 96, 20, 144]. Model-based RL techniques enable agents to leverage knowledge about the environmental model (e.g., the environmental dynamics and system objective) to improve learning speed and performance [84, 102, 75, 107].

Reinforcement learning for MDPs are able to leverage models into improving performance in a couple of ways. If agents are given an accurate model of the system, they can utilize offline learning techniques which solve the modeled MDP

directly. With online model-based RL, agents can leverage the model to guide exploration and action selection during learning towards more "interesting" regions of the search space (resulting in more efficient exploration) [102, 75]. Value iteration and policy iteration are perhaps the most popular methods of solving model-based reinforcement learning problems [125, 107].

Model-based RL algorithms either assume agents are given an accurate system model a-priori, or that agents must construct their own approximate system model through repeated interactions with the environment. In this work, we first assume that agents are given an accurate partial system model (i.e., the system reward model) a priori in order to demonstrate the benefits of our reward shaping techniques. We then extend these techniques to the case where agents must learn their own approximate system model (i.e., an approximate model of the system objective), and then use this approximate model during learning.

2.2 Multiagent Learning

Learning is often an important component of multiagent systems. Learning methods can generally be grouped into one of three main categories: *supervised*, *unsupervised*, and *reward-based* learning. Supervised learning is learning in the presence of a 'teacher', which tells an agent whether the action it took was right or wrong. 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 is simply put in an environment and learns patterns from its inputs and observations without receiving any explicit feedback. A common example of unsupervised learning is clustering: detecting potentially useful or related clusters from a set of input examples [116]. Reward based learning is often called "semi-supervised" learning: 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, namely multiagent reinforcement learning within an MDP framework.[3]

2.2.1 Considerations in Multiagent Learning

When designing a multiagent system, there are a number of key system properties that must be considered and determined by the system designer. Here we introduce a few of these properties:[4]

Credit Assignment Problem: In cooperative multiagent learning problems, credit assignment involves how to distribute rewards to each agent in the system based upon their individual contribution to the system performance. Credit assignment has two key aspects: 1) How to divvy up credit among agents based upon the actions they took individually and determine how they contributed to the system performance (structural credit assignment), and 2) How to distribute credit

[3] A comprehensive list of single and multiagent reinforcement learning algorithms can be found in [30, 125, 154].

[4] Additional information on these topics can be found in [116, 125, 153].

to agents for actions taken during previous time steps (temporal credit assignment problem) [8]. In order to achieve good performance in multiagent learning, both aspects of the credit assignment problem must be addressed [125].

Communication in Multiagent Systems: Communication is a critical element in multiagent learning. Communication can be done either *explicitly* or *implicitly*. Explicit methods involve direct agent-to-agent communication, negotiation, or information exchanging. In such cases, communication is typically limited by domain attributes. Some domains have a maximum communication distance while others have restricted data rates. Most approaches to explicit communication involve assigning a cost to communication and having agents try to coordinate in order to minimize the communication costs incurred [149]. Implicit communication is indirect and commonly occurs through environmental interactions. A few examples of implicit communication include coupling agent reward functions [8] or leaving pheromones or trails that other agents in the environment can detect and follow [89]. Tradeoffs between different types and levels of communication and system performance is highly studied in multiagent systems research.

Centralized vs. Decentralized Learning: In general, there are two extremes associated with controlling large sets of autonomous devices. The first extreme is a centralized control approach which treats the individual agents as peripherals of a single system controller[5]. This approach is advantageous in that the centralized control prevents individual agents from taking conflicting actions. However,

[5]If a multiagent system has full communication in which all agents can communicate directly with each other, it is effectively equivalent to a system with a single centralized controller [123].

there are several drawbacks and difficulties associated with centralized approaches. These approaches become prohibitively expensive to compute as the number of agents within the system increases, since the computation often increases exponentially with additional agents due to agent-to-agent interactions within the system. In addition, such approaches require the central node to receive complete "sensory" information from each node in the system (full observability), which is unrealistic in many real-world domains. An alternative is a fully decentralized approach, in which each agent in the system develops its own view of the environment and policy of acting in it. A completely decentralized approach is advantageous because it avoids single points of failure, easily adds and removes agents from the system, and is readily reconfigurable to dynamically changing system requirements. Decentralized approaches have been heavily researched in recent years due to these capabilities. Current drawbacks associated with these methods lie in the difficulty associated with designing individual-agent policies that collectively work together to optimize the joint-actions of all the agents in the system and resultant system performance.

2.2.2 From Single Agent Learning to Multiagent Learning

When extending single agent algorithms to multiagent learning, two new key issues arise: How to account for the collective action of other agents in the system and how to select actions that not only provide a direct benefit but also shape the actions of other actions in the future. The first issue is an "input" problem, in that it forces an

agent to differentiate between the potentially stochastic changes to an environment from the actions of intelligent agents and exploit this knowledge. The second issue is an "output" issue, in that it forces the agent to determine a course of actions that through its interaction with other agents influence the actions of other agents. These two issues together provide both theoretical (convergence) and practical (signal to noise in rewards) complications and render the direct application of single agent learning algorithms problematic.

To date, most work focused on the output problem, particularly when focusing on cooperative systems where agents aim to maximize a common goal [21, 47, 155]. Applications for such multiagent systems include managing air traffic [103, 136], coordinating teams of unmanned aerial vehicles [47], managing the electrical grid [87], controlling complex power plants [88], managing online digital auctions [37], and data mining [155] to name a few. However, despite their benefits, multiagent learning approaches must be improved to generalize to a larger set of coordination problems, and both explicit and implicit coordination mechanisms are two options to handle this issue.

Explicit coordination involves direct interaction and agent-to-agent communication between two or more agents within a system [6, 68, 74, 94]. Key examples of explicit coordination mechanisms include auctions, bidding, and other forms of negotiations, which have been used for the allocation of goods and resources, as well as task allocation in multiagent systems [37, 78, 91, 105, 110, 126]. Additionally, techniques involving graph-based agent dependencies and message passing (e.g., max-plus, coordination graphs) have been used in a number of domains including

controlling multiple robots as well as swarms of UAVs [24, 47, 85, 94]. However, the complexity and computational costs of explicit coordination techniques make these techniques well suited to domains with a few sophisticated agents, but not to domains where very large number of computationally limited agents must be coordinated [5, 85, 116, 150, 156, 157].

Implicit coordination techniques rely solely upon an agent's observation of the environment, other agents, and the surrounding system to make decisions. Stigmergy, which enables agents to coordinate through their interactions with the environment (e.g., leaving pheromone trails for other agents to detect and follow) is one common form of implicit coordination that has shown success in a number of applications including controlling robot exploration and UAV swarms [55, 89, 94, 116]. Another approach to implicit coordination is through reward shaping, which designs agent-specific rewards that reflect how well the agents interacted (i.e., how their joint actions benefited the system performance) [15, 29, 43]. Key examples of reward coupling include potential-based reward shaping and difference rewards which have both been shown to increase both learning speed as well as overall performance in a number of multiagent domains [14, 15, 42, 43, 103].

2.2.3 Rewards for Multiagent Learning

One of the most critical factors within multiagent learning is selecting the rewards each agent should use to learn. The first and most direct approach is to let each agent use the system reward as their individual reward. However, in many domains,

especially domains involving large numbers of agents, such a reward often leads to slow learning. This is because each agent has relatively little impact on its own reward. For instance if there were 100 agents and an agent takes an action that improves the system reward, it likely that some of the 99 other agents will take poor actions at the same time, and the agent that took a good action will not be able to observe the benefit of its own individual action and how it impacted the resultant system performance.

Another possibility for a reward is to use a local agent-specific reward that only accounts for the action of the particular agent. While with such rewards an agent can easily see the impact of its action on its reward, in most domains, the local rewards are not aligned with the system reward $G(z)$. In such domains, an agent can maximize its own local reward, but in doing so it can reduce the overall system reward. Local reward structures are primarily useful in problem domains in which the local reward can be created in such a way that they are directly aligned with the system performance. However, in many complex problem domains it is notoriously difficult to derive agent-specific reward structures that are perfectly aligned with the system objective function.

In this work, we develop novel reward mechanisms for promoting learning, coordination, and scalability within multiagent systems. In particular, we focus on designing a generalized class of reward structures which can be used across a plethora of domains to improve performance in multiagent learning. These rewards are able to maintain the benefits of global rewards $G(z)$ (e.g., being positively aligned with the objective of the system) while at the same time reaping the

benefits associated with localized rewards (e.g., the ability for agents to get clear feedback on how their individual actions contributed to the reward they received). The general term for the techniques used to create these rewards is reward shaping

2.2.3.1 Reward Shaping

Reward shaping is the practice of replacing an agent's reward function with an alternative reward that changes its learning [43, 132]. Frequently, reward shaping is used to improve system performance or to make a problem easier to solve [13, 53]. Reward shaping has been used to increase performance by speeding up convergence rates and improving coordination in problems involving reinforcement learning [13, 149]. In Q-learning, reward shaping can be represented by the following formula [43, 95]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + F(s,s') + \gamma max_{a'} Q(s',a') - Q(s,a)] \qquad (2.2)$$

where $Q(s,a)$ is the Q-value associated with the agent taking action a in state s, r is the standard reward, a' is an alternate action, s' is an alternate state, α is the learning rate, γ is the discount factor, and $F(s,s')$ is the general form of the shaping reward. As seen, the shaping reward $F(s,s')$ is an additional reward that is applied on top of the agents original reward r in order to encourage better learning [43, 95]. Reward shaping techniques (e.g. Potential-based reward shaping [43]) have been used to increase performance by speeding up convergence rates and improving

coordination in problems involving reinforcement learning [13, 43, 53, 149].

Many existing reward shaping techniques require a full system model in order for them to be computed (i.e., agents or the system designer must have access to both the transition probabilities $T(s, a, s')$ as well s the system reward model $R(s, a)$ [43, 53, 149]. Unfortunately, in complex multiagent systems, this information is not always available, rendering many reward shaping techniques (e.g., potential-based reward shaping) inapplicable to many real-world applications.

Next, we will introduce a type of reward shaping technique which was developed to both improve coordination and performance within multiagent systems, while concurrently reducing the amount of information required by agents during learning. More specifically, we introduce difference rewards, which are shaped rewards that only require agents to have knowledge of the system reward model $R(s, a)$ and eliminate the need for agents to have knowledge of the transition probabilities $T(s, a, s')$.

2.2.3.2 Factoredness and Learnability

Ideally, a reward should provide an agent with two key pieces of information: 1) How its action impacted the overall system performance, and 2) How its action impacted the reward it received. Feedback on how its own actions impacted the system performance allows agents to make decisions that are in-line with the system objective. Providing agents with feedback on how its individual actions impacted the reward it received allows the agent to adapt its actions in order to benefit both

itself and the system.

This first property has been formalized for an agent j, by defining the **degree of factoredness** (also presented in [141, 152, 151]) between the agent-reward g_j and system reward G at state z, as:

$$F_{g_j} = \frac{\sum_{z} \sum_{z'} u\left[(g_j(z) - g_j(z'))(G(z) - G(z'))\right]}{\sum_{z} \sum_{z'} 1} \tag{2.3}$$

where the states z and z' only differ in the state of agent j, and $u[x]$ is the unit step function, equal to 1 if $x > 0$. The numerator keeps track of the number of state pairs (z, z') for which the agent reward, $g_j(z) - g_j(z')$, and system reward, $G(z) - G(z')$, are aligned (have the same sign). A high degree of factoredness means that agents improving their own local reward are concurrently improving the system performance, while agents harming their local reward are also harming system performance.

The second property has been defined as **learnability**, which is the degree to which an agents reward, g_j, was impacted by its own actions as opposed to the actions of other agents. The learnability of a reward, g_j, for agent j, evaluated at z can be quantified as follows:

$$L_{g_j} = \frac{||g_j(z) - g_j(z - z_j + z'_j)||}{||g_j(z) - g_j(z' - z'_j + z_j)||} \tag{2.4}$$

where in the numerator z' differs from z only in the state of agent j, and in the denominator the state of all other agents is changed from z to z'. Intuitively, the

learnability provides a ratio between the portion of the agents reward signal that depended upon its own actions (signal), and the portion of its reward signal that depended upon the actions of all other agents (noise). The higher the learnability, the easier it is for an agent to learn an accurate mapping between its actions and its rewards.

2.2.3.3 Difference Rewards

Difference rewards are a particular type of shaped rewards which emphasize assigning credit to individual agents in the system based upon their independent contributions to the system's performance [60, 12, 59, 132, 141, 151, 13, 141, 152, 151]. Difference rewards have been shown to work well in a number of domains and conditions [7, 60, 59, 63, 13, 132]. Difference rewards are shaped rewards of the form [13]:

$$D_j \equiv G(z) - G(z_{-j} + c_j) \tag{2.5}$$

where G is the system objective, z is the complete system state vector and z_{-j} contains all the variables not affected by agent j. All the components of z that are affected by agent j are replaced with the fixed constant c_j (counterfactual action).[6] These rewards only require agents to have access to a model of the system objective G and do not require agents to have any knowledge of the system transition probabilities T, which significantly reduces the overhead requirements

[6]The only requirement of difference rewards is that agents have some approximation of the shape of the underlying system objective, G [103].

compared to existing reward shaping techniques.

Difference rewards are factored no matter what the choice of c_j, because the second term does not depend on j's actions [132]. Furthermore, they usually have far better learnability than does a team reward, because the second term of D, which removes a lot of the effect of other agents (i.e., noise) from j's reward. In many situations it is possible to use a value of c_j that is equivalent to taking agent j out of the system. This causes the second term to be independent of j (i.e. the system performance without agent j), and therefore D_j evaluates the agent's contribution to the global performance. There are two key advantages to using D_j: First, because the second term removes a significant portion of the impact of other agents in the system, it provides an agent with a "cleaner" signal than G [13, 132]. Second, because the second term does not depend on the actions of agent j, any action by agent j that improves D, also improves G (the derivatives of D and G with respect to j are the same) [13, 132].

We also consider the Expected Difference Reward (EDR) which is given by:

$$EDR_j \equiv G(z) - E_{z_j}[G(z)|z_{-j}] \tag{2.6}$$

where $E_{z_j}[G(z)|z_{-j}]$ gives the expected value of G over the possible actions of agent j. Because this term does not depend on the immediate actions of j, this reward is still aligned with G [132]. Furthermore, because it removes noise from each agent's own reward, EDR yields far better learnability than does G [132]. This noise reduction is due to the subtraction which (to a first approximation)

eliminates the impact of states that are not affected by the actions of agent j. The major difference between EDR and D is in how they handle z_j. EDR provides an estimate of agent j's impact by sampling all possible actions of agent j whereas D simply removes agent j from the system.

Any system capable of broadcasting the system performance G or passing state-vector information can be minimally modified to allow agents to independently calculate their own difference reward [132]. Although difference rewards have been demonstrated to work well within a number of domains, they are still not applicable to many real-world domains where an accurate partial system model (i.e., an accurate model of the system objective) is unavailable and the agents must operate within an initially unknown environment. In this work, we address this shortcoming by first introducing CLEAN rewards which have the same partial model requirements as difference rewards (i.e., they require an accurate model of the system objective in order to be computed) but that outperform difference rewards by accounting for exploratory action noise which is introduced in Chapter 3. We then developed techniques which enable agents to construct approximate models of the system objective and then use these approximate models to calculate their individual CLEAN rewards, resulting in improved real-world applicability over existing techniques.

Chapter 3 – CLEAN Rewards for Learning in the Presence of Exploration

In cooperative multiagent systems, coordinating the joint-actions of agents is difficult. One of the fundamental difficulties in such multiagent systems is the slow learning process where an agent not only needs to learn how to behave in a complex environment, but must also account for the actions of the other learning agents. Here, the inability of agents to distinguish the true environmental dynamics from those caused by the stochastic exploratory actions of other agents creates noise on each agent's reward signal. Under these conditions, the solution (using agents) actually becomes part of the problem. This learning noise can have unforeseen and often undesirable effects on the resultant system performance. We define such noise as *exploratory action noise* and introduce a reward structure that effectively removes such noise from each agent's reward signal. In particular, we introduce two types of Coordinated Learning without Exploratory Action Noise (CLEAN) rewards which are agent-specific rewards based on an agent estimating the counterfactual reward it would have received had it taken an alternative action. Then, in the next chapter, we empirically show that CLEAN rewards outperform agents using both traditional global rewards and shaped difference rewards in two toy domains.

3.1 Introduction

Learning in large multiagent systems is a critical area of research with applications including controlling teams of autonomous vehicles [13], managing distributed sensor networks [46, 149], and air traffic management [136]. A key difficulty of learning in such systems is that the agents in the system provide a constantly changing background in which each agent needs to learn its task. As a consequence, agents need to extract the underlying reward signal from the noise of other agents acting within the environment. This learning noise can have a significant and often detrimental impact on the resultant system performance. In this chapter, we first define *exploratory action noise* present in multiagent systems and then introduce Coordinated Learning without Exploratory Action Noise (CLEAN) rewards which designed to promote coordination while removing *exploratory action noise* from each agent's reward signal.

Currently, there are two key ways for agents to account for each other within decentralized multiagent systems: 1) agent-modeling techniques, and 2) treating agents as a part of the environment. Agent modeling techniques have been shown to work well in a number of settings, but they quickly become intractable as scaling increases [71, 114, 123]. Other issues arise when agents are treated as a part of the environment (e.g. *exploratory action noise*), and their exploratory actions are seen by other agents as stochastic environmental dynamics. Here, the inability of agents to distinguish the true environmental dynamics from those caused by the stochastic exploratory actions of other agents creates noise on each agent's reward

signal. This problem cannot simply be addressed by turning off exploration and acting greedily (this has been repeatedly shown to result in poor performance as agents always exploit their current knowledge which is frequently incomplete or inaccurate [125]). We address this by introducing CLEAN rewards which are designed to effectively remove much of the learning noise caused by agents taking exploratory actions.

The key innovation of our approach is that agents never explicitly take exploratory actions. Instead exploration is accomplished by agents privately computing a "counterfactual" reward they would have received had they taken an exploratory action. These counterfactual rewards and actions are then used to update their policies. In this way exploration for an agent is kept "private" to that agent, and does not result in noise being added to the system. Only an agent's non-exploratory action is seen by other agents. This paradigm promotes agent-to-agent coordination (agents are constantly coordinating with each others' current "best" policy) while maintaining the exploration needed during learning (private counterfactuals provide agents with rewards that approximate the impact of changing their current policies). Through utilizing "private" exploration, learning with CLEAN rewards effectively removes the *exploratory action noise* associated with learning, which simplifies the coordination problem for agents and improves scalability (Sections 3.2, 3.3, and 4.2).

The primary contributions of this chapter are to:

- separate environmental noise from noise caused by the exploratory actions of agents by defining *exploratory action noise*.

- introduce two variations of Coordinated Learning without Exploratory Action Noise (CLEAN) rewards (Section 3.3) which promote coordination and remove the *exploratory action noise* associated with multiagent learning.

The remainder of this chapter is structured as follows: Section 3.2 provides background on exploratory action noise and reward shaping. Section 3.3 introduces two variations of CLEAN rewards and discusses the benefits and drawbacks of each.

3.2 Background and Related Work

In this work, we focus on the impact one agent learning has on another agent's ability to learn. To ground our discussion, we will focus on reinforcement learning agents [125] (though our concepts naturally flow to other search and learning concepts such as evolutionary algorithms). Reinforcement learning agents observe the state of the environment, and take actions to maximize their expected future reward. A reinforcement learning problem consists of at least one agent and an environment. A reinforcement learning agent has four key elements including a *policy*, *reward function*, *value function*, and optionally an *environmental model* [125].

An agent's *policy* defines the way the agent behaves at any given time and maps the agent's perceived environmental state to the action it takes in that state [125]. These policies are initialized in an arbitrary manner, and adjusted over time as the agent learns. Each agent's policies are updated via the agent's *reward* and *value function*. An agent's *reward* function rates the performance of the agent.

The reward function provides an agent with a learning signal for actions that it took. The reward may come directly from the environment or it can be shaped to provide specific feedback to the agent. The *value function* captures the agent's internal valuate of a given action at a given time. In the limit, it converges to the true reward (possibly discounted) the agent would receive for taking an action.

3.2.1 Exploration Exploitation Dilemma

In practice, reinforcement learning techniques are used to enable an agent to autonomously learn its own control policies by repeatedly interacting with its environment [145, 158]. In this setting, the agent "learns" by continuously taking actions in an effort to learn the underlying reward associated with those actions [106, 120, 158]. For single-agent learning, it has been proven that reinforcement learning techinques such as Q-learning will converge to the optimal control policy in infinite time assuming the agent explores each possible state-action pair an infinite number of times [145, 146]. Of course in practice, this has to be achieved in finite time, and exploration strategies are implemented. That is, agents balance taking the best action (exploiting current knowledge) with querying the environment on other options (exploring). The way these two concepts are balanced is known as the exploration-exploitation dilemma [125, 158, 109].

Exploration aims to improve long term performance by uncovering actions that may provide beneficial in the future, whereas exploitation aims to maximize long term performance by assuming that the currently known values are in fact the

correct ones. It has been proven that there is no optimal exploration strategy for all situations [108, 130, 131, 142], and instead a number of different exploration strategies have been developed for agent learning [31, 106, 108]. Perhaps the simplest exploration strategy is the ϵ-greedy strategy, where agents choose to exploit their current knowledge $(1-\epsilon)$ percent of the time, and choose to explore ϵ percent of the time [125, 145, 146]. Unfortunately, the computational costs of ϵ-greedy exploration techniques increase exponentially with the size of the system [130, 131], meaning that in many cases, more intelligent and efficient exploration strategies need to be used. To that note, more complex strategies such as Boltzman exploration, which weights exploration to more frequently explore actions that are currently believed to be "better" [125]. Additional exploration techniques with varying requirements and computational requirements exist such as using confidence bounds [21], Kalman filter dynamics [120], linear perceptrons [142], and Bayesian sampling [19]. This issue has been explored to great lengths in the context of reinforcement learning [2, 3, 19, 23, 31, 99, 109, 127, 128, 98, 97, 80, 124, 148, 158].

3.2.2 Exploratory Action Noise

In cooperative multiagent systems, coordinating the joint actions of agents is difficult [43, 45, 51]. One of the fundamental difficulties in such multiagent systems is the slow learning process where an agent may not only need to learn how to behave in a complex environment, but may also need to account for the actions of the other learning agents [16, 116, 125]. Here, the inability of agents to distinguish

the true environmental dynamics from those caused by the stochastic exploratory actions of other agents creates noise on each agent's reward signal [124, 21, 143]. Under these conditions, the solution (learning agents) actually becomes part of the problem.

There are two broad approaches to handling the stochasticity due to agents, which involve modeling other agents, and treating agents as a part of the environment. Extensive work has been done with agent modeling, though such approaches are best suited to small multiagent systems where agents repeatedly interact with the same agents[100, 123, 33]. Treating agents as a part of the environment on the other hand has been shown to significantly impact both learning convergence and learning performance [27, 38, 70]. There have been several approaches to address this though, such as the "Win Or Learn Fast" (WOLF) framework which adjusts the learning rate of individual agents based upon the stochasticity of the environment due to other agents (effectively controlling how rapidly agents change their policies) [27, 28, 41]. Although these techniques have been shown to work well in many situations, they do not explicitly allow the agents to remove the noise on their reward signals associated with the exploratory actions of agents during learning. As a consequence, though they provide good results, they do not eliminate the effects of learning noise on the resultant system performance.

It is common for agents treat each other as part of the environment such that the exploratory actions of other agents are treated as stochastic environmental noise. However, under such assumptions, the agents are unable to distinguish when their peers are taking purposeful actions, from when they are taking random

exploratory actions. Here, agents are frequently adapting their policies to better coordinate with the random exploratory actions of other agents, meaning that agents will end up learning to bias their policies such that they actually depend upon the exploratory actions of other agents in order to perform well. This means that agents learning optimal policies in the presence of exploration may not be optimal once learning is complete and exploration is turned off (Figure 1.1). In this setting, the agents' inability to distinguish between true environmental dynamics and dynamics caused by the exploratory actions of other agents means that the agents themselves (the solution) actually end up becoming part of the problem (added complexity due to stochastic learning noise).

We define such noise as *exploratory action noise*, which can be defined as the portion of an agent's learning signal that is impacted by the exploratory actions of other agents. We quantify *exploratory action noise* as follows:

$$N_{\epsilon,i} = \frac{|g_i(\mathbf{a}) - g_i(\mathbf{a} - \mathbf{a}_\epsilon))|}{|g_i(\mathbf{a})|} \tag{3.1}$$

where $N_{\epsilon,i}$ is the value of the *exploratory action noise* for agent i, g_i is agent i's reward function, $\mathbf{a}$ is the joint-action vector for all agents in the system, and $\mathbf{a}_\epsilon$ is the subset of the joint-action vector containing all elements of $\mathbf{a}$ that were exploratory actions. Intuitively, if the system exploration rate is $\epsilon = 1.0$, then $N_\epsilon = 1.0$ since the entire action vector contains exploratory actions. Similarly, if $\epsilon = 0$ then no agents are taking exploratory actions and the value of $N_\epsilon = 0$, meaning there is no exploratory action noise present.

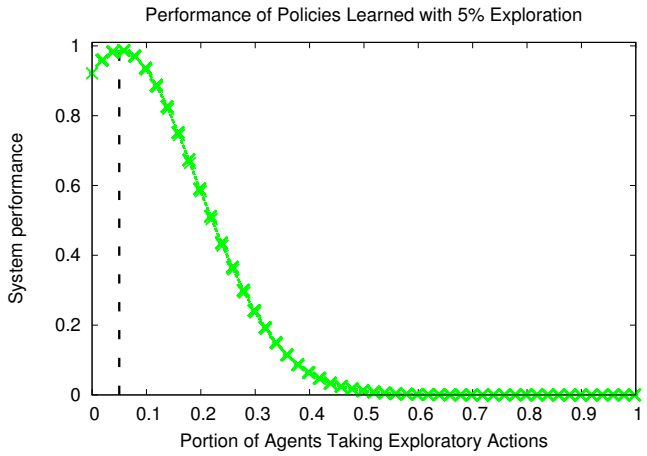

Figure 3.1: ϵ-greedy reinforcement learning agents were trained for 1000 episodes and then learning was turned off and their policies were fixed. We then test the performance of the learned policies under different amounts of exploration actions (we replace the policies of some agents with random exploratory policies). In these cases, agents perform better in the presence of exploration than they do without exploration. In particular, these policies perform the best when the portion of agents taking exploratory actions is near the value of ϵ used during learning ($\epsilon = 0.05$ in this case). This is because the agents learn to depend upon the exploratory actions of other agents. Although the exploration-exploitation trade-off has been extensively studied throughout the multiagent learning literature, relatively little work has been done to address the biasing issues associated with agents learning to depend upon the exploratory actions of other agents.

As an example, we consider the case of a set of agents learning in the Gaussian Squeeze Domain used in this work (Section 4.1). In these experiments, a set of learning agents are trained for 1000 episodes and then learning is turned off and the set of learned policies are followed (Figure 1.1). As seen, after 1000 episodes of learning, once the learning is turned off and the policies are fixed (meaning that there is no longer any stochastic exploratory actions), agents that had learned good policies in the presence of exploration actually perform worse when the exploration

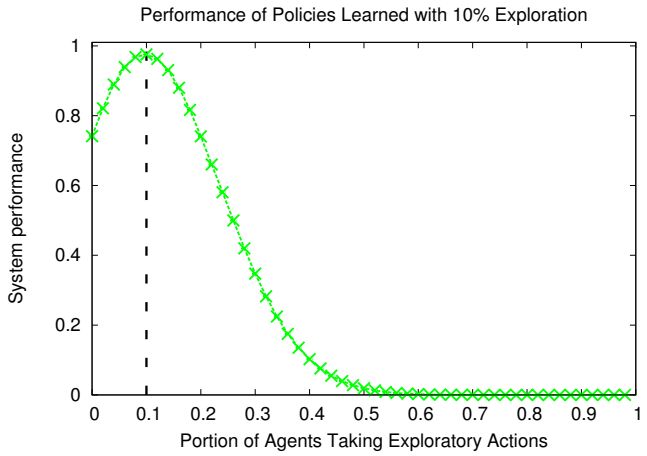

Figure 3.2: Here, agents learned for 1000 episodes and then learning was turned off and their policies were fixed. We then test the performance of the learned policies under different amounts of exploration actions (we replace the policies of some agents with random exploratory policies). In these cases, agents perform better in the presence of exploration than they do without exploration. In particular, these policies perform the best when the portion of agents taking exploratory actions is near the value of ϵ used during learning ($\epsilon = 0.10$ in this case). This is because the agents learn to depend upon the exploratory actions of other agents. Although the exploration-exploitation trade-off has been extensively studied throughout the multiagent learning literature, relatively little work has been done to address the biasing issues associated with agents learning to depend upon the exploratory actions of other agents.

is removed. This loss of performance is counterintuitive since in general, we would assume that removing exploration noise would improve the performance of agents. However, as seen here, this is not always the case. This is because these agents have actually learned to depend upon the exploratory actions of the other agents as a part of their solution.

An additional representation of the dependency on *exploratory action noise* can be observed in Figures 3.1 and 3.2. Here, we train agents for 1000 episodes

as we did in Figure 1.1 under 5% and 10% exploration, respectively. We then take the learned policies of the agents and test their performance under varying degrees of exploration. In each case, agents learned a set of policies, such that the joint-policy was $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$, where Π is the learned joint-policy, π_i is the individual policy learned by agent i, and N is the total number of agents. In Figures 3.1-3.2, we replace various portions of the policies π_i with random policies. By replacing the "true" policies with random policies, we simulated the presence of exploratory action noise in the system. As seen, performance tends to be best when a portion of the agents that is equivalent to the training exploration rate are replaced with random policies. This is because during learning, approximately ϵ portion of the agents were taking random actions at any given time and the learned joint-policy learned to account for these actions and incorporate them into their solution. It is important to note that the learned policy is not always worse once exploration is turned off and these examples are meant to demonstrate the potential for exploration to interfere with learning in multiagent systems. In general, even if the learned policies are better without exploration, it will still frequently be the case that exploration noise inhibited the overall learning performance of the system. CLEAN rewards are designed to avoid dependencies upon the exploratory actions of other agents in the system by performing exploration via off-policy counterfactual actions which do not create explicit environmental noise.

3.3 CLEAN Rewards

Designing intelligent behavior for autonomous systems becomes more difficult as the complexity of these autonomous systems increases. This difficulty becomes especially acute in multiagent systems, where even the simplest of agents can combine to form complex behavior. Learning algorithms give a promising solution to this problem. By automatically exploring the control policy space of the system to find good behaviors, the burden of creating intelligent agents is taken off of the system designer. At their core, all of these learning algorithms use an elegant exploration loop, where an agent explores an action, receives feedback about the performance of the action, and updates its control policy based this action. Exploration is key to this process, as we need to explore actions we know little about to see if they are in fact good actions. However in both single agent and multiagent systems this leads to the "exploration / exploitation" dilemma: In complex systems, most random actions will be bad, therefore exploratory actions in action spaces we know little about are likely to be bad, yet it is most important to explore action spaces we know little about because these spaces may contain the solution we are looking for.

This core "exploration vs. exploitation" issue is a fundamental design challenge for any learning system and is not addressed in this work, but leads us directly to our focus. Here we focus on a second and important exploration dilemma unique to multiagent systems: How can an agent effectively learn a good policy with respect to other agents' nominal behavior, when the other agents are not exhibiting their

nominal behavior, since they also need to explore for their learning process? In other words, agents need to learn in the context of what other agents are doing, yet the exploration "noise" of the other agents can fundamentally change the nature of this context. Having multiple agents learn together inherently creates a complex self-organizing system. Traditionally the outcome of such a system is unpredictable as it depends on the complex interactions the objective functions agents are trying to maximize and the concurrent exploration done to achieve this maximization. These issues create a real challenge to a system designer, who has to design reward functions and exploration policies that will lead to a satisfactory outcome. By addressing multiagent exploration and completing our following objectives, we will allow designers to create more stable multiagent systems, that are easier to design and have higher performance.

In this section we introduce Coordinated Learning without Exploratory Action Noise (CLEAN) rewards which were developed to address issues arising from learning noise caused by exploration in order to promote learning, coordination, and scalability in multiagent systems.[1] These rewards utilize reward updates based upon privatized counterfactual actions which allow agents to approximate rewards associated with actions that were not explicitly taken.[2] The key requirements of CLEAN rewards is that agents have an accurate model of the underlying system objective, G, and that the agents in the system follow their current target policies. Traditionally, following target policies has been shown to perform poorly due

[1]CLEAN rewards were designed using the theory of collectives [11, 13, 136, 141].

[2]Similar rewards based upon counterfactual actions known as "difference rewards" have been shown to work well in a number of domains and conditions [11, 13, 136].

to a lack of exploration, however, CLEAN rewards address this shortcoming via privatizing agent exploration (Equations 3.2 and 3.3).

CLEAN rewards are structured in such a way that they promote implicit coordination that leads to good system performance (agents improving their own local reward are concurrently improving the system performance, while agents harming their local reward are also harming system performance) and are also designed to address the structural credit assignment by providing each agent with specific feedback on how its own actions impacted the reward it received (CLEAN rewards are sensitive to the actions of the individual agent). Additionally, in order to remove exploratory action noise, agents learning with CLEAN rewards do not perform traditional explicit exploratory actions within the environment, instead agents all greedily follow their current target policies at each time step. Exploration comes from private counterfactual actions c_i, which provide each agent with an approximation of the reward it would have received had it not followed its target policy, but instead had taken some alternative action c_i.

3.3.1 CLEAN 1: $\mathcal{C}_{1,i}$

Difference rewards (Section 2.2.3.3) have been shown to work well in a number of domains and conditions [136, 141], however, they do not account for the learning noise caused by exploratory actions of agents in the learning process which can have unforeseen effects learning. CLEAN rewards maintain the strengths of difference rewards, while at the same time removing noise associated with the exploratory

actions of agents in the system (Section 3.3). Our first variation of CLEAN rewards is defined as follows:

$$\mathcal{C}_{1,i} \equiv G(z_T - z_{T,i} + c_i) - G(z_T - z_{T,i} + c_i') \tag{3.2}$$

where $\mathcal{C}_{1,i}$ is the CLEAN reward of agent i, z_T is the system state vector that results from the agents following their current target policies, $z_{T,i}$ is the actual action of agent i, c_i and c_i' are two counterfactual actions of agent i (i.e. alternative actions agent i could have taken instead of following its greedy target policy action $z_{T,i}$), and G is the system objective. These CLEAN rewards replace the contribution of the agent's target action $z_{T,i}$ with two different counterfactual actions c_i and c_i', in the first and second terms, respectively. Here, the agent approximates the reward it would have received if it would have taken actions c_i and c_i'. Then, the agent compares the counterfactual rewards associated with each action, and provides the agent with a reward for action c_i based upon the difference between the two approximations (Equation 3.2). It is frequently possible to select a counterfactual action c_i' that is equivalent to removing agent i from the system. In this setting, the CLEAN reward would provide a reward approximation that tells the agent its contribution to the system. Hence, when choosing a counterfactual for the second term that is equivalent to removing the agent from the system, the $\mathcal{C}_{1,i}$ reward will provide the agent with a positive reward if the counterfactual action c_i would have been beneficial to the system, and a negative reward if the counterfactual action c_i would have been harmful to the system. With the $\mathcal{C}_{1,i}$ rewards in this work,

the counterfactual c_i is chosen "randomly" each episode of learning according to an exploration strategy (effectively enabling the agent to calculate counterfactuals for various "potential actions" and resulting in "privatized exploration"), and the constant c_i' is selected in such a way that it is equivalent to removing agent i from the system so the second term is effectively "the system without agent i" (although alternate values for c_i' would also be valid - we discuss another way it could be selected next).

3.3.2 CLEAN 2: $\mathcal{C}_{2,i}$

Although $\mathcal{C}_{1,i}$ rewards promote good agent-specific feedback, agents must perform two separate calculations per reward update (one calculation associated with c_i and one calculation associated with c_i'). Next, we will introduce an alternative CLEAN reward which requires each agent to perform only one counterfactual action calculation per reward update. Instead of computing CLEAN rewards based upon two separate counterfactual actions c_i and c_i', if the agent uses its original target action, $z_{T,i}$, in the second term, it can avoid the calculation of the second term for the counterfactual action c_i'. In this case, an agent's CLEAN reward can be represented as follows:

$$\mathcal{C}_{2,i} \equiv G(z_T - z_{T,i} + c_i) - G(z_T) \tag{3.3}$$

where $\mathcal{C}_{2,i}$ is the CLEAN reward of agent i, z_T is the system state vector that results from the agents following their current target policies, $z_{T,i}$ is the action of agent i, c_i is a counterfactual action of agent i (i.e. an alternative action agent i could have taken instead of following its target policy), and G is the system objective. Here, agents directly compare the system reward with their own counterfactual action c_i (first term of $\mathcal{C}_{2,i}$ to the system reward associated with following following its current target policy action (second term of $\mathcal{C}_{2,i}$). Intuitively, this gives the agent a reward that represents how the system would have performed had it not followed its target policy, but instead had taken some counterfactual action c_i. In addition to providing faster computation (agents no longer have to compute the second term associated with c_i'), the $\mathcal{C}_{2,i}$ reward also provides a reward signal that constantly seeks to improve the agent's target policy directly.

Chapter 4 – CLEAN Rewards in Toy Problems

In this chapter, we implement the CLEAN rewards introduced in the previous chapter in two toy domains in order to demonstrate their performance benefits. More specifically, we implement CLEAN rewards within a congestion problem (i.e., the Gaussian Squeeze Domain) and a combinatorial optimization problem (i.e., the Defect Combination Problem) to show their performance under varying conditions. As a performance benchmark, we compare the performance of CLEAN rewards against both traditional team-based "global" rewards, as well as against existing state-of-the-art reward shaping techniques (e.g., difference rewards).

4.1 Experimental Domains

There are two domains used in this chapter, the first is a congestion domain called the Gaussian Squeeze Domain (GSD) and the second is a combinatorial optimization domain called the Defect Combination Problem.

4.1.1 The Gaussian Squeeze Domain

This domain assumes that there exists a set of agents which each contribute to a system objective, and the agents are attempting to learn to optimize the system objective (i.e. agents are attempting to coordinate their joint-action to optimize

for the 'capacity' in the Gaussian system objective) . The objective function for the domain is as follows:

$$G = xe^{\frac{-(x-\mu)^2}{\sigma^2}} \tag{4.1}$$

where x is the cumulative sum of the actions of agents (i.e., $x = \sum_i x_i$, where x_i is the "contribution" action of agent i), μ is the mean of the system objective's Gaussian (effectively the target "x" that the agents are aiming for), σ is the standard deviation of the system objective's Gaussian. Here, the goal of the agents is to choose their individual actions x_i in such a way that the sum of their individual actions is to optimize Equation 4.1. Here, each agent has 10 actions ranging in participation value from zero to nine. The GSD is a congestion domain, where adjusting the variance changes the coordination complexity for agents within the system. The lower the variance, the higher the coupling of agents' joint actions.

4.1.2 The Defect Combination Problem

Many real world sensing applications require large sets of disparate sensing devices to coordinate their actions in order to collectively optimize their network attenuation, coverage areas, and sensing schedules [46, 111, 149]. In this domain, a set of sensing devices must coordinate their sensing schedules in order to optimize their aggregated attenuation. This is the Defect Combination Problem (DCP) domain introduced in [34]. This problem assumes that there exists a set of imperfect sensors, $\mathbf{X}$, which have constant attenuations due to manufacturing defects or im-

perfections. Each of the sensors, x_i, has an associated attenuation, ζ_i, (which can be positive or negative) in its reading, such that if it is taking a measurement of A (actual value) it measures $A + \zeta_i$ where ζ_i is the device's individual error. The problem then becomes how to best choose a subset of the $\mathbf{X}$ sensors that minimizes the aggregated attenuation of the combined readings:

$$G = \frac{\left| \sum_{i=1}^{N} n_i \zeta_i \right|}{\sum_{i=1}^{N} n_i} \tag{4.2}$$

where G is the aggregated attenuation of the combined sensor readings, ζ_i is the attenuation of a particular sensor i, N is the number of sensors, and $n_i \in \{0, 1\}$ based upon whether the sensor chooses to be "on" or "off".

This is an NP-complete optimization problem [34, 136] and simply choosing the single sensor with the best attenuation is an inadequate solution, as is choosing the best K sensors ($1 \leq K \leq N$). To illustrate this, consider the case where there are 6 sensing devices whose attenuations are $\zeta_1 = -0.19$, $\zeta_2 = 0.54$, $\zeta_3 = 0.1$, $\zeta_4 = -0.14$, $\zeta_5 = -0.05$, and $\zeta_6 = 0.21$. Choosing only the best sensor ζ_5 would yield an aggregated attenuation of $|0.05|$, while choosing sensors ζ_3, ζ_4, and ζ_5 would yield an aggregated attenuation of $|0.03|$, which is better than the single best sensing device ζ_5 alone. This is still not the optimal solution in this 6 sensor case however, as combining sensors ζ_1 and ζ_6 results in an aggregated attenuation of $|0.01|$. In this problem, individual sensors acting independently without coordinating their

actions can drastically decrease the system performance. Consider the case where sensors ζ_1 and ζ_6 are turned on in conjunction with sensor ζ_2, the aggregated attenuation jumps to from $|0.01|$ to $|0.18|$. Finding good solutions requires a great deal of coordination between sensors, as any one sensor can heavily impact the system performance.

4.1.3 CLEAN Rewards for the DCP

Both CLEAN reward structures $\mathcal{C}_{1,i}$ and $\mathcal{C}_{2,i}$ can be derived for any multiagent domain. Here, we derive CLEAN rewards for agents in the Defect Combination Problem (DCP) used in this work (CLEAN rewards can similarly be derived for the GSD domain). First, we derived $\mathcal{C}_{1,i}$ by combining Equations 4.2 and 3.2 as follows:

$$\mathcal{C}_{1,i} \equiv G(z_T - z_{T,i} + c_i) - G(z_T - z_{T,i} + c_i') \tag{4.3}$$

$$\mathcal{C}_{1,i} = \frac{\sum_{j=1}^{N} n_j\zeta_j - n_i\zeta_i + c_i\zeta_i}{\sum_{j=1}^{N} n_j - n_i + c_i} - \frac{\sum_{j=1}^{N} n_j\zeta_j - n_i\zeta_i + c_i\zeta_i}{\sum_{j=1}^{N} n_j - n_i + c_i} \tag{4.4}$$

As seen, many of the terms in the first and second terms cancel out. In particular, many terms which are not directly impacted by agent i cancel out. This property is important in addressing the structural credit assignment problem, as this cancellation allows agent i to more clearly see how its own actions impacted its reward as opposed to the actions of other agents. Further simplification yields the

following:

$$\mathcal{C}_{1,i} = \begin{cases} 0, & \text{if } c_i = 0 \text{ },c_i' = 0, \\ \frac{\sum_{i \neq j}^{N} n_i \zeta_i}{\sum_{i \neq j}^{N} n_i} - \frac{\sum_{i \neq j}^{N} n_i \zeta_i + c_i' \zeta_i}{\sum_{i \neq j}^{N} n_i + c_i'}, & c_i = 0, c_i' = 1 \\ \frac{\sum_{i \neq j}^{N} n_i \zeta_i + c_i \zeta_i}{\sum_{i \neq j}^{N} n_i + c_i} - \frac{\sum_{i \neq j}^{N} n_i \zeta_i}{\sum_{i \neq j}^{N} n_i}, & c_i = 1, c_i' = 0 \\ 0, & c_i = 1, c_i' = 1 \end{cases}$$

and similarly, we derived $\mathcal{C}_{2,i}$ by combining Equations 4.2 and 3.3 to yield the following:

$$\mathcal{C}_{2,i} = \begin{cases} 0, & c_i = 0, z_{T,i} = 0 \\ \frac{\sum_{i \neq j}^{N} n_i \zeta_i}{\sum_{i \neq j}^{N} n_i} - \frac{\sum_{i \neq j}^{N} n_i \zeta_i + z_{T,i} \zeta_i}{\sum_{i \neq j}^{N} n_i + z_{T,i}}, & c_i = 0, z_{T,i} = 1 \\ \frac{\sum_{i \neq j}^{N} n_i \zeta_i + c_i \zeta_i}{\sum_{i \neq j}^{N} n_i + c_i} - \frac{\sum_{i \neq j}^{N} n_i \zeta_i}{\sum_{i \neq j}^{N} n_i}, & c_i = 1, z_{T,i} = 0 \\ 0, & c_i = 1, z_{T,i} = 1 \end{cases}$$

where in the above equations, $\mathcal{C}_{1,i}$ and $\mathcal{C}_{2,i}$ are the two variations of CLEAN rewards for agent i in the Defect Combination Problem, ζ_i is the attenuation of agent i's sensing device, N is the total number of agents, and the variables c_i, c_f, n_j, and n_i are all indicator variables with value 0 corresponding to a sensor turned off and 1 corresponding to a sensor turned on (Section 4.1). CLEAN rewards could similarly be derived for the Gaussian Squeeze Domain, but are excluded here for brevity.

4.2 Results

This section includes the experimental results for both the Gaussian Squeeze Domain and the Defect Combination Problem Domain. All experiments consist of $r = 100$ statistical runs, all Q-tables are initialized to zero, and the error in the mean $\sigma/\sqrt{r}$ was plotted in all experiments although it is so small that it is not visible in many experiments (all results were statistically significant as verified by a t-test with $p = 0.05$ for all experiments). In the GSD the learning rate and exploration rate were set to $\alpha = 0.10$ and $\epsilon = 0.10$ and in the DCP they were set to $\alpha = 0.01$ and $\epsilon = 0.01$, unless otherwise stated.

There are four types of reward structures used in this work: difference rewards (Equation 2.5) denoted by D, global rewards denoted by G, and two variations of CLEAN rewards denoted CLEAN_1 (Equation 3.2) and CLEAN_2 (Equation 3.3). In the following experiments, G, D, CLEAN_1, and CLEAN_2 represent the *online* performance of agents during learning in the presence of exploration and G_OFF and D_OFF represent the *offline* performance of the underlying policy being learned by agents (the performance of the joint-policy learned up to that point if learning and exploration were turned off, see Figure 1.1).[1] Here, the difference in the performance of the policies G and D as compared to G_OFF and D_OFF can be directly contributed to *exploratory action noise*.

[1]The online and offline performance of CLEAN rewards are identical since agents explore via off-policy counterfactual actions while continually following their target policies.

4.2.1 The Gaussian Squeeze Domain

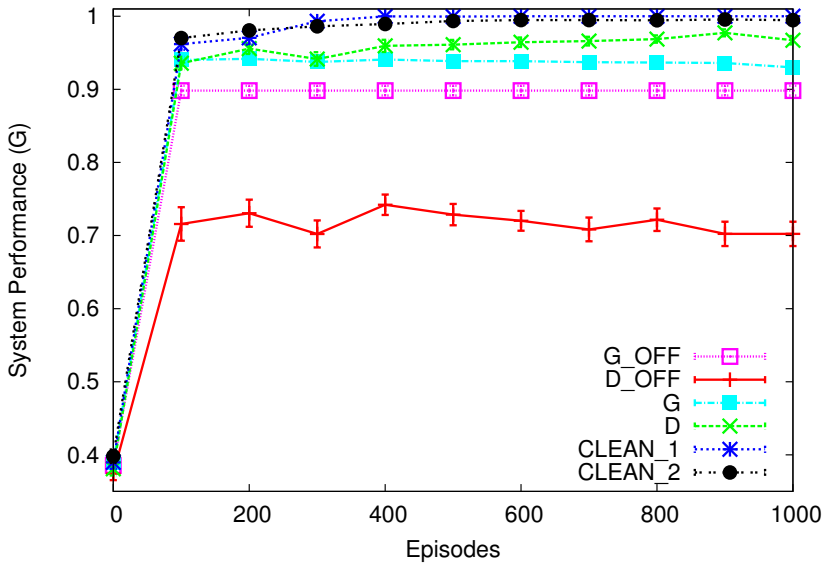

Figure 4.1: $N = 500$ agents learning in the Gaussian Squeeze Domain with $\mu = 0.80N$ and $\sigma = 0.80N$, and 10% exploration. As seen, agents using G and D experience significant decreases between their online and offline performance. This is because agents learn policies that depend upon the presence of *exploratory action noise*. CLEAN rewards are robust to such noise and have improved performance.

In Figure 4.1, we see the results for 500 agents learning in the Gaussian Squeeze Domain with $\mu = 0.80N$, $\sigma = 0.80N$, and $\epsilon = 0.10$. This experiment is meant to show the potential for agents to learn to depend upon the exploratory actions of other agents. As seen, learning with all reward structures perform well during learning (G, D, CLEAN_1, CLEAN_2), however the actual policies being learned by agents are drastically different for agents using global and difference rewards (G_OFF and D_OFF). Once learning is turned off and agents using global and difference rewards follow their fixed policies, agents using global rewards experience

a drop in performance of 3%-5%, while agents using difference rewards experience a drop in performance of approximately 30% (Figure 4.1).

Next, we performed a set of experiments where we proportionally scaled the number of agents, the mean, and the variance (Figure 4.2). This provides further insight into some situations where the agents can learn to depend upon the exploratory actions of other agents. In Figure 4.2, the offline performance of agents using global and difference rewards (G_OFF and D_OFF) are almost always worse than the online performance where exploratory actions were present (G, D). It is important to note that this is not always the case, for example Figure 4.2 shows that in these experiments, when 100 agents are present G_OFF slightly outperforms G. However, the purpose of these results is to show that *exploratory action noise* does impact learning in multiagent systems in frequently unforeseeable and unpredictable ways. As seen in both of these examples, CLEAN rewards perform well in the presence of *exploratory action noise*.

Now that we have demonstrated that *exploratory action noise* does play a role in learning in multiagent systems and that CLEAN rewards learn policies that are robust to *exploratory action noise*, we want to demonstrate the performance benefits of CLEAN rewards when the problem becomes more complex. In this congestion domain, as the variance decreases, the agents become more coupled and the *exploratory action noise* becomes increasingly important. In Figure 4.3, we maintain a fixed mean and variance, while scaling the number of agents in the system. This effectively increases the problem complexity and the importance of *exploratory action noise* (more agents taking exploratory actions have more impact

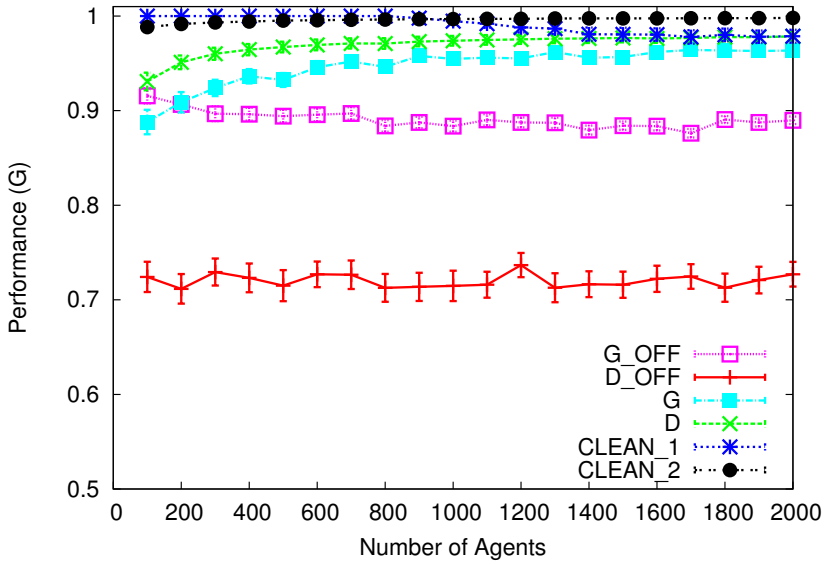

Figure 4.2: Proportional Agent Scaling with $\mu = 0.80N$ and $\sigma = 0.80N$, where N is the number of agents. This is a low-complexity setting in the Gaussian Squeeze Domain (high variance). Again, the offline performance for agents using G and D is frequently worse than the online performance. This is because the underlying policies learned by agents relied upon the exploratory actions of the other agents in the system.

upon the system performance in low variance settings). As seen in this setting, CLEAN rewards are much more robust when the problem complexity is increased (more agents and higher congestion) and the exploratory actions of agents play a larger role in system performance.

4.2.2 The Defect Combination Problem

In the following experiments, we demonstrate the performance of agents using CLEAN rewards in the Defect Combination Problem (DCP) domain, which is a

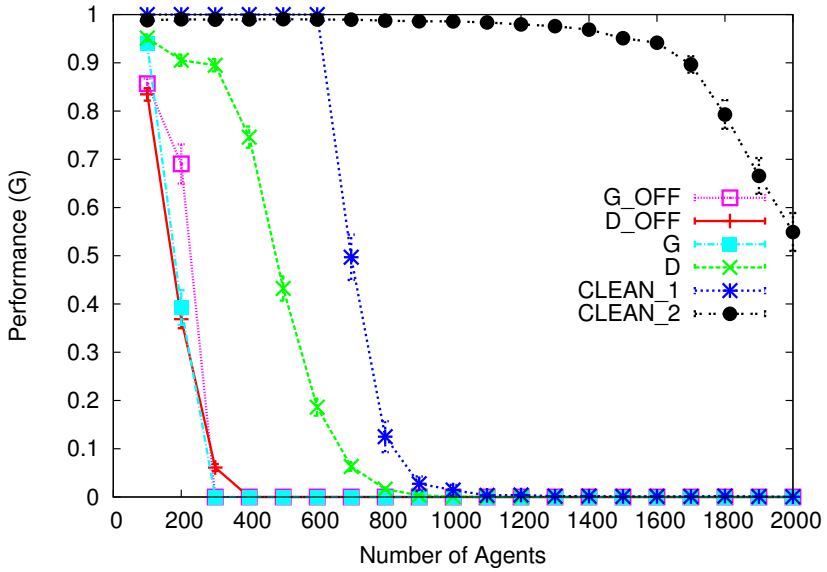

Figure 4.3: Scaling the number of agents with $\mu = 100$ and $\sigma = 100$. As the number of agents increases, the coupling between agents increases and the problem becomes more difficult and the *exploratory action noise* has more of an effect on system performance. As seen, CLEAN rewards are more robust than other rewards.

combinatorial optimization problem. In Figure 4.4, we see the performance of 300 agents in the DCP. Again, CLEAN rewards outperform agents learning with both global and difference rewards, respectively. The key difference in performance between D and CLEAN can be attributed to the filtering of *exploratory action noise* (CLEAN rewards and difference rewards are similar in that they both attempt to provide agent-specific feedback that promotes implicit coordination, however, CLEAN rewards have the added advantage that they simplify learning by removing *exploratory action noise* - Sections 3.2 and 3.3). Due to the combinatorial aspects of the DCP, the offline performance tends to be greater than the online performance for agents using global and difference rewards, this is because in the DCP the

exploratory actions of a single agent can drastically change the system performance (Section 4.1.2). However, the presence of exploratory actions during the learning process still impacts the performance of the policies learned by these agents, as agents using global and difference rewards actively attempt to coordinate with agents that are routinely taking random exploratory actions. CLEAN rewards address this issue by having agents follow their current target policies and providing agents with rewards based upon counterfactual actions, which effectively allows the agents to explore without causing exploration noise on each others' learning signals (Section 3.3). As seen, agents using CLEAN rewards are able to achieve aggregated attenuations that are nearly an order of magnitude better than other techniques (Figure 4.4).

Next, we analyzed the performance of agents in the Defect Combination Problem for scaling the number of agents. As seen in Figure 4.5 agents using global rewards perform poorly as scaling increases. Similarly, agents using difference rewards perform well with up to 300 agents, but then as the number of agents is increased, their performance decreases. A key cause of this decrease in performance is the inability of difference rewards to account for the exploratory actions of agents. Although proportionally the same number of agents are exploring (e.g. 1% exploration), as the number of agents in the system increases, there are more agents exploring. The learning noise caused by these agents can significantly inhibit learning, especially in combinatorial optimization problems. In this setting, the fact that agents are constantly attempting to coordinate their policies to account for each others' exploratory actions significantly inhibits performance. CLEAN

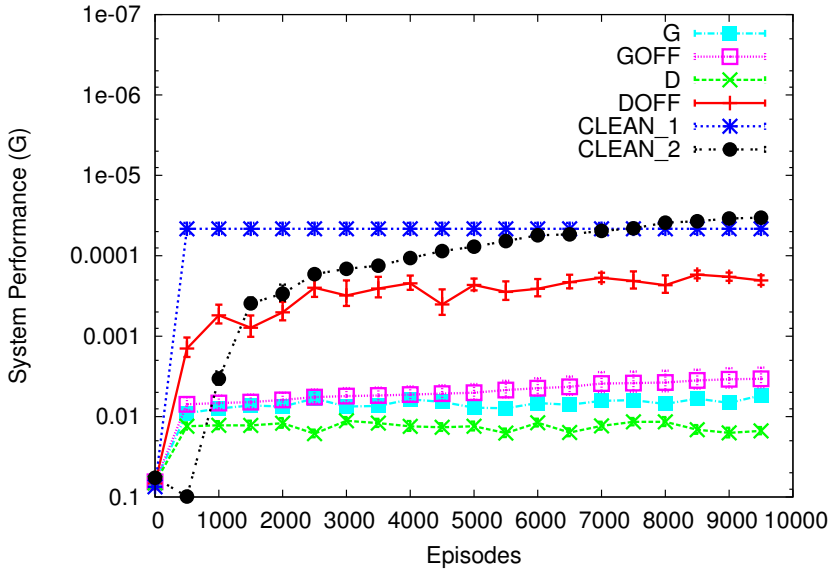

Figure 4.4: The Defect Combination Problem with 300 agents. Exploratory actions can be especially damaging in this domain, as a single agent can have a drastic impact on the overall system performance (Section 4.1.2). CLEAN rewards maintain exploration while avoiding the coordination issues that arise when agents attempt to coordinate in the presence of exploration (Sections 3.2 and 3.3).

rewards address this shortcoming by filtering out learning noise caused by agent exploration, resulting in up to three orders of magnitude better performance when scaling up to 2000 sensing agents (Figure 4.5).

4.3 Discussion

In multiagent learning (e.g., multiagent reinforcement learning), a set of agents continually interact with their environment (e.g., take actions) in order to learn how to behave within that environment (i.e., learn how to optimize their individual objectives as a part of the larger system). In particular, agents begin with no prior

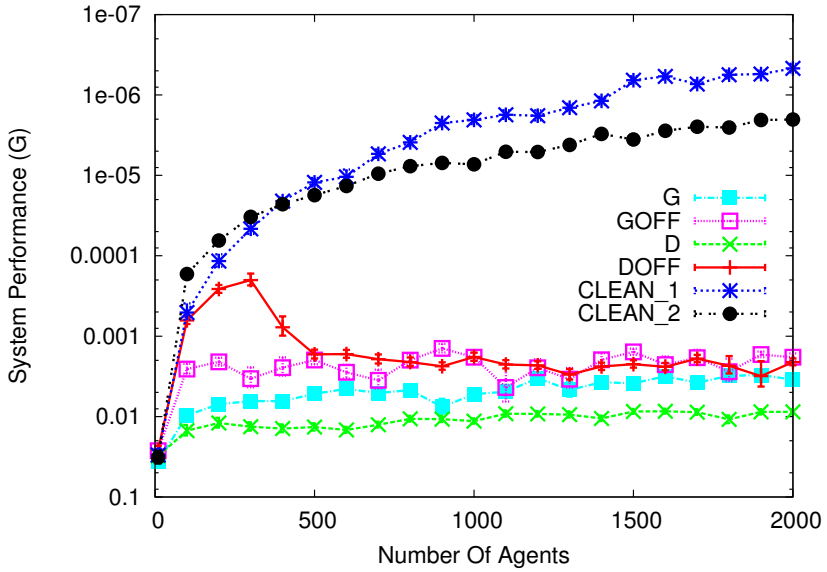

Figure 4.5: The Defect Combination Problem scaling the number of agents from 10 to 2000. As seen, CLEAN rewards are highly scalable, outperforming other methods by up to three orders of magnitude for high levels of scaling.

knowledge or experience with the environment, and iteratively learn how to behave (i.e., gain knowledge of how to interact with the environment) through a process of trial and error. Here, the agents continually strike a balance between *exploiting* the knowledge they have already gained (e.g., taking actions that currently they believe will lead to good performance) and *exploring* new actions which may improve their knowledge of the system and eventually lead to better performance. It is well-known throughout the multiagent and machine learning communities that learning agents must balance these two behaviors and this has become well-known as the "exploration-exploitation" tradeoff in learning [86, 125].

While there has been a lot of research involving the exploration-exploitation tradeoff, relatively little work has been done to directly address the impact of learn-

ing noise caused by the exploratory actions of agents. Typically exploration noise is handled at the same time as the core exploration / exploitation problem: The system is "annealed", where exploration rates are reduced throughout learning or otherwise tinkered with to achieve better performance. However, this approach confounds the two exploration dilemmas and forces agents to have low exploration towards the end of learning. This may be when the agents can benefit from exploring the most, since the system has stabilized. In addition, the outcome of the system now depends on a complex self-organizing process based upon not only the system dynamics and the objective functions, but also the changing exploration levels. A better solution is to eliminate exploration noise at all stages of learning, while retaining the key benefits of the exploration process.

Unfortunately, in a multiagent learning setting, the exploratory actions of individual agents actually make it more difficult for other agents to learn (i.e., agents actually confuse each other when they take exploratory actions). This is because in this setting, agents treat each other as a part of the environment such that the exploratory actions of other agents are treated as stochastic environmental noise. Under these assumptions, the agents are unable to distinguish when their peers are taking purposeful actions (e.g., exploiting their current knowledge and taking actions they "know" lead to good performance) from when they are taking random exploratory actions (e.g., randomly sampling the action space to attempt to improve their system knowledge). Here, agents are frequently adapting their control policies to better coordinate with the stochastic exploratory actions of other agents, meaning that agents will end up learning to bias their policies such that they actu-

ally depend upon the exploratory actions of other agents in order to perform well. There has been a lot of research involving the exploration-exploitation tradeoff within the multiagent learning literature. However, relatively little work has been done to directly address the impact of learning noise caused by the exploratory actions of agents.

In this work, we first showed the potential impact of such exploratory action noise on learning, demonstrating that exploratory actions can cause agents to bias their policies to depend upon the exploratory actions of others, which can lead to suboptimal learning. We defined this learning noise as *exploratory action noise*. We then introduced CLEAN rewards, which are shaped rewards designed specifically to promote coordination and scalability in multiagent systems by addressing both the structural credit assignment problem, as well as the *exploratory action noise* caused by agent exploration. Then we provided empirical results demonstrating the performance of CLEAN rewards in two multiagent domains including a Gaussian Squeeze Domain (congestion domain) and a Defect Combination Problem (combinatorial optimization domain). We showed that CLEAN rewards are highly robust to exploration and scaling, significantly outperforming both global rewards and difference rewards in the two domains used in this chapter.

Although CLEAN rewards address the issue of exploratory action noise, they still have two key shortcomings. First, agents using these rewards are only able to perform a single reward update per episode of learning, which significantly slows down the learning speed of the system. Second, these rewards still require agents to have access to an accurate model of the system objective in order to compute

their individual CLEAN rewards. In the coming chapters, we will address each of these shortcomings sequentially. We exploit a natural extension of CLEAN rewards which leverages the concept of privatized counterfactual exploration to enable agents to perform multiple reward updates during each time step (resulting in significantly increased learning speed). Then, we improve the real-world applicability of CLEAN rewards by enabling agents to construct their own approximate model of the system objective when an accurate system model is unavailable a priori. Agents then use their approximate models to calculate their individual CLEAN rewards. This extension significantly improves the real-world applicability of CLEAN rewards by enabling them to be calculated in initially unknown environments.

Chapter 5 – Batch-CLEAN Rewards for More Efficient Learning

Learning within multiagent systems such that agents jointly achieve a common goal represents a complex coordination problem. This problem is exacerbated with scaling as the joint policy space that agents must search through grows exponentially with respect to the number of agents in the system. This exponential increase in the policy space renders many traditional learning based techniques (including the CLEAN rewards introduced in the previous chapters) inadequate for many domains due to relatively slow learning updates (e.g., agents receive only a single reward update per time step and may be unable to adapt to rapidly changing environmental conditions in a timely manner). To address this we introduce Batch Coordinated Learning without Exploratory Action Noise (Batch-CLEAN) rewards which allow agents to calculate multiple reward updates concurrently by providing agents with rewards for each of its available action selections during each episode of learning by leveraging the concept of privatized exploration to enable agents to calculate their CLEAN rewards associated with several counterfactual actions at a time. Enabling agents to calculate reward updates for multiple potential actions during each episode will result in improved learning speed.

5.1 Introduction

Traditional online learning methods for agent-based systems provide agents with a single reward update per atomic learning experience. However, in large multiagent systems such learning updates can be prohibitively slow. More specifically, the environment may change faster than the agents are able to adapt their policies due to limited information, resulting in poor performance. To address this problem, we leverage the properties of private exploration and CLEAN rewards introduced in the previous chapters to provide agents with reward updates for multiple actions during each episode of learning. More specifically, we introduce Batch-CLEAN rewards by extending CLEAN rewards to include batch reward updates during each episode. The concept of updating rewards for actions not taken is present within the literature, for example, with learning automata the probability of taking a particular action may change based upon similar actions' results [139, 93, 143]. However, this work instead centers around rewards that explicitly aim to quantify the reward an agent would have received had it taken an alternate action [139]. We explore the application of these rewards to dynamic domains where the rapidly changing conditions put a premium on learning quickly.

In this chapter, we propose a multiagent learning approach that significantly improves both learning speed (with respect to atomic learning experiences) and performance by allowing an agent to update its estimate of the rewards (e.g. value function in reinforcement learning) for all its available actions, not just the action immediately taken by the agent. In particular, we introduce Batch-CLEAN

rewards which are designed to promote coordination by making efficient use of experience data by providing each agent with multiple rewards and value updates per atomic experience. Batch-CLEAN rewards are calculated based upon counterfactual actions (actions not explicitly taken), which enables agents to calculate approximate reward values for multiple 'potential actions' during each atomic learning experience. Since these rewards are based upon counterfactual actions that the agent does not actually take, each agent is able to calculate approximate rewards and update the corresponding value associated with each of their potential actions during each time step of learning, resulting in more information per atomic learning experience

In this chapter, we provide background information on existing online multiagent learning techniques and introduce Batch-CLEAN rewards. In the next chapter, we will demonstrate the performance of Batch-CLEAN rewards in two real-world domains as compared to global rewards, difference rewards, and standard CLEAN rewards previously introduced.

5.2 Background and Related Work

Typically agents must address both the temporal credit assignment problem (how to assign a reward received at the end of a sequence of actions to each of the actions) and the structural credit assignment problem (how to assign credit to a particular agent at the end of a multiagent task) [11, 69, 71, 125, 143, 147]. These two problems have been addressed together with learning sequences of actions

for multiagent systems [32, 38, 64, 122]. Here, the learning needs of agents are adjusted to account for their presence in a larger system [13, 65]. Although learning sequences of actions has led to significant advances in multiagent systems, they are principally based upon the iterative process of an agent sampling a single action, receiving an evaluation for the action, and updating its value estimate for taking that action in the given state. Such an approach is effective, but is typically slow to converge, particularly in dynamic environments [13]. We explore the concept of agents learning from actions they do not take by estimating the rewards they would have received had they taken those actions, providing multiple reward updates. These counterfactual rewards are estimated using the theory developed for structural credit assignment [11, 13, 136].

Agent-based learning methods can be generally classified into two key categories: online learning algorithms and offline learning algorithms [45, 49, 51, 54, 72, 79, 121, 125]. In this chapter we utilize online reinforcement learning algorithms which are well suited for multiagent learning as agents learn from taking sequential actions within an environment [72, 82]. The main criticism of online reinforcement learning methods is their inefficiency in the way they use experience data [26]. This is because agents make just one incremental learning update for each piece of atomic experience data, and then discard the data [72]. Such incremental updates have the advantage of requiring little computation and memory, in exchange requiring a lot of data [72]. In many agent domains (e.g. physical robots or complex real-time simulation environments) it is more expensive to gather training data than it is to perform additional computations and data analysis [72]. In such do-

mains, the sheer time, tedium, or labor involved in gathering training experiences could be the overriding concern [72]. We show that online learning which performs multiple reward calculations per atomic learning experience can lead to significant performance gains, with fewer atomic learning episodes.

Here, we address a key factor that is a known issue with multiagent learning, which has to do with the speed with which a solution is found. Multiagent learning focuses on learning to coordinate the actions of individual agents such that they jointly achieve a common objective [50, 73, 74, 116, 125]. Due to the stochastic nature of such problems, agents need to learn to adapt their policies quickly in response to changing environmental dynamics. This can be difficult for a number of reasons including the temporal (how to assign credit for a particular action in a sequence of actions) and structural (how to assign credit to one agent based on the actions of all the agents) credit assignment problems [9, 16, 137], as well as the need for adequate data (information from interacting with the environment) [16, 72, 125]. Traditional learning methods for agent-based systems provide agents with a single reward update per interacting with the environment. However, in large multiagent systems such learning updates can be prohibitively slow [64, 122, 127, 128, 136]. More specifically, the environment may change faster than the agents are able to adapt their policies due to limited information, resulting in poor performance [60, 129]. A potential solution is to provide agents with multiple value updates based on one interaction with the environment using the theory developed for counterfactual rewards and structural credit assignment [16, 137, 136]. The concept of updating rewards for actions not taken is present within the literature,

for example, with learning automata the probability of taking a particular action may change based upon similar actions' results [93, 139, 143].

In this work we utilize reinforcement learning algorithms which are well suited for multiagent learning as agents learn from taking sequential actions with an environment [72, 82, 116, 125]. One of the key drawbacks of online reinforcement learning methods is their inefficiency in the way they use data (a drawback that becomes much more pronounced in a multiagent setting) [26]. This is because agents make just one incremental learning update for each interaction with the environment and then discard the just collected data [54, 65, 72, 79, 125]. Such incremental updates have the advantage of requiring little computation and memory, in exchange requiring a large number of interactions with the environment [72, 125]. In many agent domains (e.g. physical robots or complex real-time simulation environments) it is more expensive to gather training data by interacting with the environment that it is to perform additional computations and data analysis [45, 49, 51, 54, 72]. In such domains, the sheer time, tedium, or labor involved in gathering training experiences could be the overriding concern [50, 72, 73]. Our aim is to show that online learning which performs multiple reward calculations per learning experience can lead to significant performance gains, while requiring fewer actual learning episodes.

5.3 Batch-CLEAN Rewards

In this section we introduce two variations of Batch Coordinated Learning without Exploratory Action Noise (Batch-CLEAN) rewards which were developed to promote learning and scalability in multiagent systems. These rewards have three key benefits: i) they increase the amount of information each agent gets per episode of atomic learning experience by providing agents with reward approximations for multiple actions (agents approximate the reward they would have received for taking multiple alternate counterfactual actions individually), ii) they promote coordination by addressing the structural credit assignment problem (agents are assigned feedback reflective of their individual actions' contribution to the system performance), iii) they remove learning noise caused by the stochastic exploration of other agents.

Instead of calculating a single reward update based upon the action the agent explicitly took (as global and difference rewards do), Batch-CLEAN rewards calculate reward updates based upon a counterfactual action. Batch-CLEAN rewards leverage this property, enabling each agent to calculate reward calculations for multiple counterfactual actions during each atomic learning experience (as compared to a single reward update with standard CLEAN rewards). The mathematics that allow the computation of difference rewards also enable the calculation of counterfactual action based CLEAN rewards, and therefore Batch-CLEAN rewards [11, 13, 136]. Such rewards based upon counterfactual actions make the assumption that agents have some approximation of the system objective G.

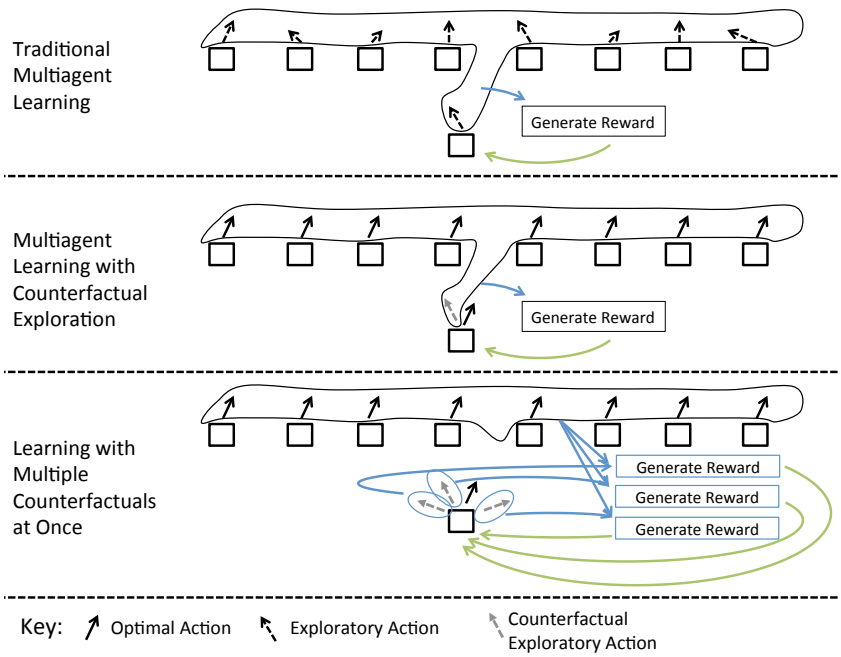

Figure 5.1: Traditionally all agents take exploratory actions and each agent receives a reward that is dependent on its action and the actions of other agents (top). The exploration of other agents causes noise on the rewards. We propose having each agent take its non-exploratory action in public, and in private take a counterfactual exploratory action, and receive a reward for this counterfactual action (middle). Multiple counterfactual exploratory actions can be taken at the same time (bottom).

Previous work involving counterfactual action based rewards has included difference rewards [11, 13, 136] and CLEAN rewards which were introduced previously in this work. Though both of these rewards have been shown to provide a reward that is tuned to an individual agent's actions, they are still based upon an agent iteratively sampling its actions over a potentially large number of atomic learning experiences. In many domains (e.g. physical robots, expensive simulations [72]), such experience can be expensive. In this section, we extend CLEAN rewards to

include multiple reward updates per atomic learning experience by leveraging the theory developed for counterfactual action updates and structural credit assignment [11, 13, 136].

5.3.1 Batch-CLEAN Rewards Based on CLEAN 1

Standard CLEAN rewards (introduced in previous chapters) addressed the structural credit assignment problem by providing each agent with a learning signal that filters off the impact of other agents in the system using a differencing technique (e.g. Equations 3.2 and 3.3). Additionally, these rewards require agents to follow their target policies, which means that agents always follow their current 'best' policy. This removes much of the learning noise associated with exploratory actions in multiagent learning (exploration is done privately, which improves learning and coordination. Although CLEAN rewards provide tangible performance benefits, agents using CLEAN rewards still have significantly inhibited learning speed as they are only able to calculate a single reward update per episode of learning.

Batch-CLEAN rewards address this shortcoming by exploiting the fact that these rewards are calculated based upon privatized counterfactual actions, enabling each agent to calculate multiple reward updates per atomic learning experience. Using Batch-CLEAN rewards, each agent calculates the set of rewards $\overline{CB1}_j = \{CB1_j(c_{j,k})\}_{k=1}^{k\leq n}$ for its entire set of counterfactual actions during each episode of learning, where $c_{j,k}$ is the counterfactual associated with action k for agent j, and there are n total actions. Here, each individual reward $CB1_j(c_{j,k})$ is calculated as

follows:

$$CB1_j(c_{a_{j,k}}) \equiv G(z_{T,-j} + c_{a_{j,k}}) - G(z_{T,-j} + c_{a_{j,k'}}) \tag{5.1}$$

where $CB1_j(c_{a_{j,k}})$ is the reward that the agent uses to update its value associated with action, $a_{j,k}$, based upon the counterfactual constant, $c_{a_{j,k}}$, used to calculate that reward, $z_{T,-j}$ is the system state vector that results from agents greedily following their current target policies without the portions dependent upon the actions of agent j, G is the system objective function, and $c_{a_{j,k'}}$ is an alternate counterfactual action. Agents using Batch-CLEAN rewards perform one reward calculation for each of the n actions, meaning these rewards scale linearly with the number of actions.

Each reward $CB1_j(c_{a_{j,k}})$ compares the relative benefit associated with two different counterfactual actions $c_{a_{j,k}}$ and $c_{a_{j,k'}}$, respectively. These rewards compare the difference in the approximate system performance, G, with counterfactual actions $c_{a_{j,k}}$ and $c_{a_{j,k'}}$, respectively. Thus, if $c_{a_{j,k}}$ is more beneficial to the system performance than $c_{a_{j,k'}}$, the agent receives a positive reward update for action $a_{j,k}$, and a negative reward otherwise. It is frequently possible to choose a counterfactual $c_{a_{j,k'}}$ that is equivalent to removing the agent from the second term's performance calculation (as is done in this work). In this case, such a reward provides an agent with an approximate contribution of its counterfactual action, $c_{a_{j,k}}$, to the system performance. These rewards promote coordination in that any action good for the agent is also good for the system and they promote learning by effectively "filtering" off much of the impact of other agents via the differencing

of the first and second terms with respect to the actions of other agents. Here the impact of many of the agents in the system is removed from the agents learning signal. Additionally, it is frequently possible to select a counterfactual action $c_{a_{j,k'}}$ that is equivalent to removing the agent out of the system. This causes the reward to yield the approximate contribution of agent j taking counterfactual, $c_{j,k}$, to the system.

5.3.2 Batch-CLEAN Rewards Based on CLEAN 2

Although Batch-CLEAN rewards of the $CB1_j$ type promote good learning, they require two separate calculations to be computed for each individual reward calculation in the set $\overline{CB1}_j = \{CB1_j(c_{a_{j,k}})\}_{k=1}^{k\leq n}$. This is because the agents are forced to calculate a counterfactual based value for both the first and second terms of the reward calculation. If instead of calculating a counterfactual action based value for the second term, the agent simply used the actual system performance (which is based upon the action the agent actually took), the agent would no longer need to calculate the second counterfactual term (reducing the computational expense for these rewards by approximately half).

To address this we introduce a second variation of Batch-CLEAN rewards which have reduced computation requirements. Here, each agent calculates the set of rewards $\overline{CB2}_j = \{CB2_j(c_{a_{j,k}})\}_{k=1}^{k\leq n}$, where each reward is computed as follows:

$$CB2_j(c_{a_{j,k}}) \equiv G(z_T - z_{T,j} + c_{a_{j,k}}) - G(z_T) \tag{5.2}$$

where $CB2_j(c_{a_{j,k}})$ is the Batch-CLEAN reward of agent j that depends upon the counterfactual action of agent j denoted by $c_{a_{j,k}}$, $G(z_T - z_{T,j} + c_{a_{j,k}})$ estimates the performance of the system had agent j taken the counterfactual action $c_{a_{j,k}}$ instead of the action it actually took, and $G(z_T)$ is the actual system performance. In this setting, the agent no longer needs to compute a counterfactual value of the second term, meaning it only needs to perform half of the calculations compared to the first type of Batch-CLEAN rewards.

Since all agents using Batch-CLEAN rewards are always following their current target (greedy) policies, $CB2_j$ rewards provide an agent with feedback on whether or not the proposed counterfactual action $c_{a_{j,k}}$ is more or less beneficial to the system than its current "best" action. Thus, $CB2_j(c_{a_{j,k}})$ rewards encourage agents to adjust their own individual actions to directly improve the current target joint-policy for agents within the system (agents acting to improve their individual Batch-CLEAN rewards simultaneously act to increase the system performance G).

Chapter 6 – CLEAN and Batch-CLEAN in Real-World Domains

In the previous chapters, we introduced CLEAN rewards to improve coordination within multiagent systems and demonstrated their performance compared to existing state-of-the-art techniques. Then, in the previous chapter, we proposed an extension of these techniques to improve overall learning speed by enabling agents to perform multiple reward updates per learning episode. More specifically, we introduced Batch-CLEAN which extends CLEAN rewards by exploiting the concept of privatized exploration to enable agents to calculate batch updates for multiple actions during each time step. In this chapter we implement Batch-CLEAN rewards in two real-world domains and demonstrate their performance benefits in comparison to standard CLEAN rewards, global rewards, and difference rewards.

6.1 Experimental Domains

In this chapter we utilize two real-world domains in order to compare the tangible benefits of CLEAN rewards and Batch-CLEAN rewards. The first is a UAV Communication Network (UAVCN) domain originally introduced in [7]. The second is a CubeSat Coordination Domain (CCD) based upon the idea of satellite resource sharing and fractionated satellite systems which has previously been introduced in [60]. We borrowed domain parameter values from those found for these domains

within the literature unless otherwise specified [7, 59, 60].

6.1.1 UAV Communication Network Domain

In the near future, solar UAVs will play critical roles in the military, industrial, scientific, and academic communities [101, 113, 115]. These devices have seemingly limitless applications including communications, reconnaissance missions, space launch platforms, and wireless power beaming [57, 101]. Recent missions including NASA's Pathfinder-Plus and QinetiQ's Zephyr (which remained airborne for over two weeks nonstop) have advanced the state of the art in solar powered UAVs, taking them from limited mission life and endurance to the point they can remain operational for weeks at a time [57, 104]. As a result of the increasing capabilities and availability of these devices coupled with their falling costs, a plethora of novel domains and applications will emerge to utilize the newly developed technological capabilities of these platforms [101].

As we progress into the information age, communication becomes an increasingly critical component of every day life. Today, cellular phones, laptops, hand held computers, and other wireless electronic devices have changed the way we see and interact with the world. At the core of these advancements is a well designed wireless communication network, which handles the workload and facilitates information sharing between devices connected to the network. Current networks rely on a series of radio towers to facilitate this information sharing work load. Traditional towers have worked well to date, but they have several key drawbacks:

1. They are expensive to build.
2. They are expensive to maintain.
3. They have limited communication due to obstructions (cannot communicate "around" obstructions).
4. They have static placement (holes in coverage areas).

Here, we focus on a subset of this domain where there is a set of UAVs that are flying at fixed locations (flying in small circles) for long periods of time (perhaps months or years) and are transmitting data to a set of customers below (see Figure 6.1). UAVs have an advantage in sending data from high altitude in that they can have line-of-sight communication to many customers. In addition by virtue of being overhead, such UAVs can focus on what areas of the surface they will project most of their signal power to, allowing for better coverage.

In this domain, each UAV can communicate to multiple customers. In addition communication is done over a shared channel (over the same frequency band) analogous to the way WiFi networks transmit data. Using a shared channel allows the system to be very adhoc, where UAVs can come and go, and can decide whether or not to participate in the system without any need for channel arbitration. Note that for simplicity we only look at the download problem, where UAVs are sending information down to customers. Also we make no assumptions on how the UAVs get their data feeds. We believe that this half of the problem is the most important, as typical internet use tends to be dominated by download traffic. Although the

uplink problem is fairly similar as long as it is done on a different channel than the downlink.

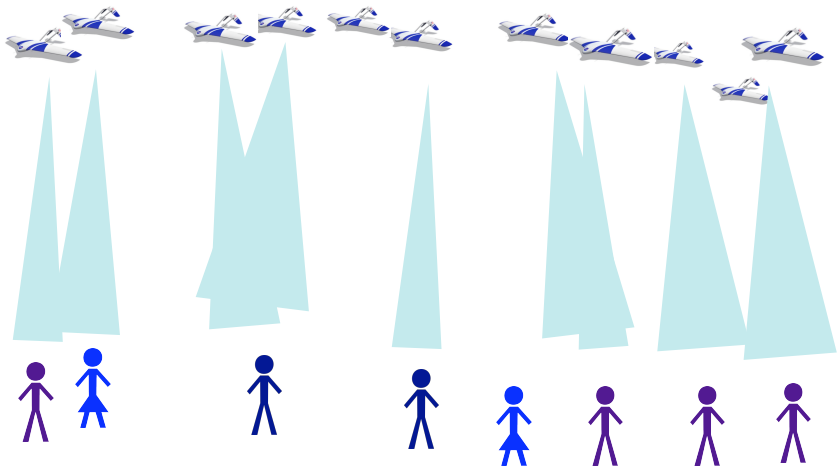

Figure 6.1: UAV Communications. A set of UAVs at high altitude transmit data to a set of customers on ground over a single communication channel. The task of the system is to maximize average bitrate customers receive. Multiple UAVs may communicate to single customer. A UAV communicates to at most one customer.

6.1.1.1 Signal Dynamics

We assume that the UAVs are all at similar altitudes and communicate through directional antennas pointed towards the ground. The amount of area on the ground that is covered by the UAV is determined by the gain of its antenna. Antennas with low gain, transmit over a wider area, but within that area the strength of the signal is lower (see Figure 6.2). Antennas with high gain, have more signal power in the center of their area, but transmit over a smaller area. The maximum signal received from a UAV is proportional to the inverse square of

the gain radius for the antenna:

$$S_j^{max} = aP_j/r_j^2 \tag{6.1}$$

where a is a constant, P_j is the power transmitted from UAV j, and r_j is the signal half-power radius for UAV j. S_j^{max} is the amount of signal received directly at the center of the transmission. Further from the center, the amount of signal received decreases exponentially according to the signal radius:

$$S_{i,j} = e^{-b\frac{r_{i,j}}{r_j}} S_j^{max} \tag{6.2}$$

where b is a constant and $r_{i,j}$ is the distance from customer i and the center of UAV j's transmission. The noise received by customer i is simply the sum of the signal from all the UAVs it is not communicating with:

$$N_i = \sum_{j \notin J_i} S_{i,j} + k \ , \tag{6.3}$$

where J_i is the set of UAVs customer i is communicating with and k is a constant for background noise. The maximum communication rate for customer i can then be estimated from the signal-to-noise ratio using Shannon's law:

$$C_{i,j} = B \log_2(1.0 + S_{i,j}/N_i) \ , \tag{6.4}$$

where B is the bandwidth of the channel in Hz [1]. The total data rate for customer i is the sum of the data rates for each UAV the customer is communicating with:

$$C_i = \sum_{j \in J_i} C_{i,j} \ . \tag{6.5}$$

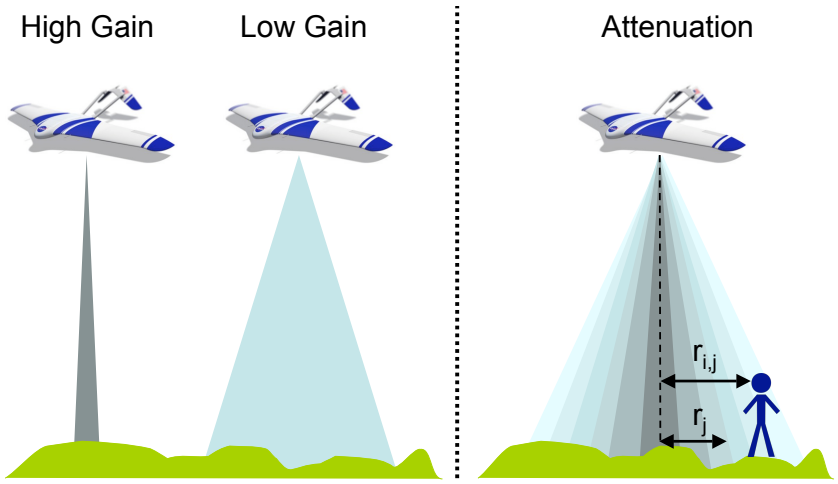

Figure 6.2: Signal Dynamics. UAVs with high-gain antennas throw a strong signal over a small area. UAVs with low-gain antennas throw weaker signal over larger area (Left). The strength of the signal depends on how far the customer is away from the center of the signal cone (Right).

[1] For simplicity, certain factors, such as relative inverse square distance signal attenuation are ignored that were determined to have little impact on performance.

6.1.1.2 System Evaluation Function and Agents

The objective of this problem is to maximize the average data rate of each customer:

$$G = \frac{1}{n} \sum_{i=1}^{n} C_i \ , \tag{6.6}$$

where there are n customers, and G is the system evaluation function. Combining equation 6.1, 6.2, 6.3, 6.4, 6.5, we obtain:

$$G = \frac{1}{n} \sum_{i=1} \sum_{j \in J_i} B \log_2 \left(1.0 + \frac{\frac{aP_j}{r_j^2} e^{-b\frac{r_{i,j}}{r_j}}}{\sum_{j \notin J_i} \frac{aP_j}{r_j^2} e^{-b\frac{r_{i,j}}{r_j}} + k} \right) \ , \tag{6.7}$$

putting our global objective in terms of our control variables: UAV power level, P_j, and indirectly, $r_{i,j}$, through the orientation of the UAV.

These controls allow us to change the signal-to-noise characteristics at different locations on the ground. However, this is a difficult problem as increasing the signal for one customer may increase the noise for another. It is especially difficult, since we want this communication network to be adhoc, where it is controlled in a distributed way: UAVs are entering and leaving the system, and some UAVs fail to cooperate or operate correctly. Fortunately, reinforcement learning algorithms and multiagent techniques are a natural match to this problem.

There are many possible agent definitions and controls for the UAVs in our domain, including altitude, antenna gain, power levels and antenna angle. Here

we focus on the last two: adjusting the power level P_j and orientation (the direction the transmitter points to) of each UAV (see Figure 6.3). We control each of these actions through agents. The solution to the full problem consists of the power level and orientation values for all the UAVs. However to simplify the problem, we break the task into a multiagent system, where a single agent controls both power level and orientation for each UAV. To perform control, each agent makes discrete actions. For adjusting power, the action is scaled exponentially to the action:

$$P_j = Pe^{z_j^p} \ , \tag{6.8}$$

where P is the base power and z_j^p is the action of the agent for UAV j controlling its power. To control orientation, an agent chooses one of nine directions: either straight down, or one of eight cardinal directions around the UAV. The angle of the pointing is fixed so that the center of the new orientation is moved a distance of r what it would have been if it had pointed strait down (see Figure 6.3 - right).

6.1.2 CubeSat Coordination Domain

Currently, satellites are very expensive resources that need to be coddled carefully. The costs of satellite missions can exceed billions of dollars, with teams of engineers, managers and scientists working together to extract all the information they can out of these missions. These missions are carefully planned and orchestrated by large institutions over a period of many years. However, in the near

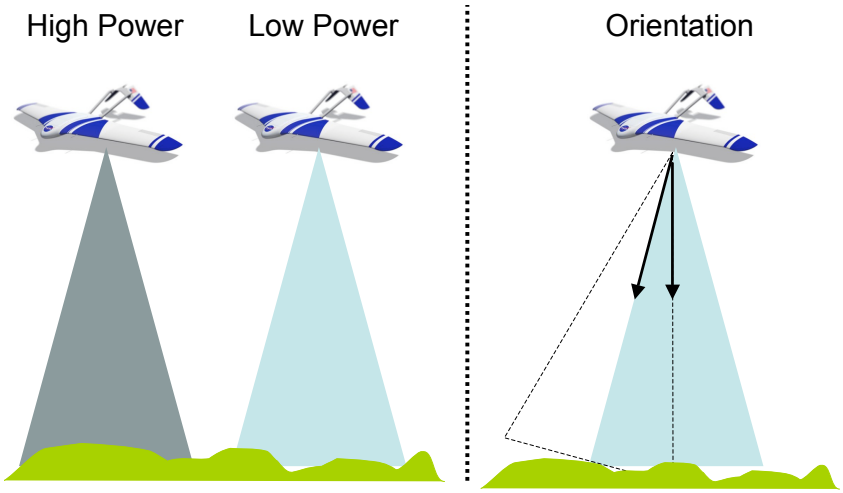

Figure 6.3: Agent Actions. An agent can choose power level of UAV within certain range. An agent can also choose orientation of antenna. The Agent must choose power levels and orientations to balance giving more signal to their customers and less noise to other customers.

future this traditional satellite paradigm could change dramatically with the introduction of very small satellites known as "CubeSats." The number of CubeSats will dramatically increase due to reduced costs coming from platform standardization, availability of COTS (commercial off-the-shelf) parts and reduced launching costs [112, 118]. These satellites will have numerous capabilities, including in situ measurements of the thermosphere, interferometry, communication and Earth observation [90]. Collaborative networks of CubeSats offer mission capabilities that are impractical for larger satellite platforms due to cost restrictions, including simultaneous in situ measurements of multiple locations in space and temporally separated measurements of precise points in space[117, 81]. In addition, they offer lower cost and increased robustness compared to traditional satellites due to system reconfigurability [22]. In addition, networking clusters of CubeSats together

in order to boost performance is becoming a popular concept, similar to computer clusters[1, 56]. However, while having numerous advantage, making effective use out of large numbers of heterogenous CubeSats is a difficult problem.

As an example, consider an instance where a small community needs to observe the realtime progress of a local forest fire. There are many aspects of this fire that can be observed from orbit, including fire intensity, distribution, and movement. A few dozen observations would be useful, but with diminishing returns beyond this number. Currently, making these observations is difficult and expensive due to the limited number of satellites. However, in the future, there may be tens of thousands of tiny CubeSats able to make these observations. How can this small community in an economical way take advantage of these resources?

A straight forward solution to this problem is a centralized satellite resource broker. Under this scenario, our small community would register its fire observation needs to the broker, and the broker would try to find the resources, trading off the costs and benefits of all the other requests that were registered. While this method is attractive for networks of large satellites, there are three main difficulties this centralized system might have with a large, but disorganized collection of CubeSats: 1) CubeSats are likely to be owned by many different countries and institutions that may not trust having their resources used by a centralized resource broker, 2) CubeSats will be in unpredictable states of repair and may be owned by institutions unable to make reliable commitments, 3) There may be so many CubeSats (perhaps millions), that a centralized system could simply not scale efficiently.

As an alternative to a centralized solution, the community could buy observations directly from the owners of the CubeSats. For this process to be effective, the community needs to do two primary things, 1) Buy the appropriate number of observations taking into account the unreliability of CubeSats, 2) Buy the observations at the lowest possible price. For these two things to happen, the reliability and expected cost of each of the CubeSats needs to be modeled so that an appropriate combination of request for observations can be made. While taking all these considerations into account would ordinarily be difficult for a small institution or community, an agent based system can help by modeling the satellites behavior and evolving policies to maximize value.

6.1.2.1 Decentralized Agent Solution

We propose to have a decentralized solution to this problem, through the use of an "agent" intermediary. But first we have to decide what an agent is in this domain.

What is an agent?

There are numerous ways agents can be used and defined. Here we explore a few alternative types of agents:

1. **Trivial**: Just pass information between CubeSats and customer.
2. **Owned by Customer**: Every customer has its own agent buying observations for that customer.

3. **Owned by CubeSat**: Every CubeSat has its own agent selling observation for that CubeSat.

4. **Independent**: One agent per CubeSat, buying observations for customer.

In the first definition, the agent is a simple intermediary. While this solution may work for a very sophisticated customer, in general it does not solve the problem of how a customer can buy an appropriate set of observations at a low price. In the second definition, each customer has an intelligent agent that tries to make these purchases for the customer. However, this solution has similar limitations to the trivial agents, as the agent would have to be sophisticated enough to know the properties of the thousands of CubeSats in existence and come up with a policy satisfying the customer's demands at a low price. In the third definition, the task of the agent is much simpler. It knows all the properties of the single CubeSat that owns it, and knows at what price points it can sell its observations for. However, the issue with this approach is that it can be very inefficient, since agents trying to maximize revenue for its CubeSat may try to sell observations that are not valuable to the customer.

Here we focus on the final definition for an agent, where *agents are independent*, there is one agent per CubeSat and the task of the agents is to buy an appropriate set of observations for a given customer. With this definition, the requirements of an agent is relatively simple. It needs to model the capabilities, reliability and price point of only a single CubeSat. Then when a customer makes an observational request to a set of agents, the agents coordinate to purchase an appropriate set of

observations. This agent model has a number of advantages:

1. CubeSats with any price structure can participate.
2. Unreliable CubeSat can participate.
3. CubeSat owner can choose not to participate on case by case basis.
4. Customers can choose not to buy resources from particular CubeSats.
5. Agents can scale with number of CubeSats.

With independent agents, CubeSats of all types can participate. An agent can simply decide not to use its CubeSat if it is inappropriate for the task. The most difficult task for an agent is to coordinate effectively with other agents. In this paper, we will focus on this coordination problem, and how policies can be evolved that allow agents to coordinate well. Note that this paper does not cover the broader case where there are multiple customers at the same time. However we believe that a similar mapping could be made in this case, where the agents are trying to maximize a larger overall utility over all customers.

6.1.2.2 Proposed Solution

We propose a multiagent solution, where independent agents help a consumer of satellite resources, buy an appropriate combination of resources at low cost (see Figure 6.4). In this algorithm an agent is assigned to every CubeSat, and is responsible for making a monetary bid to its CubeSat for its observation. The

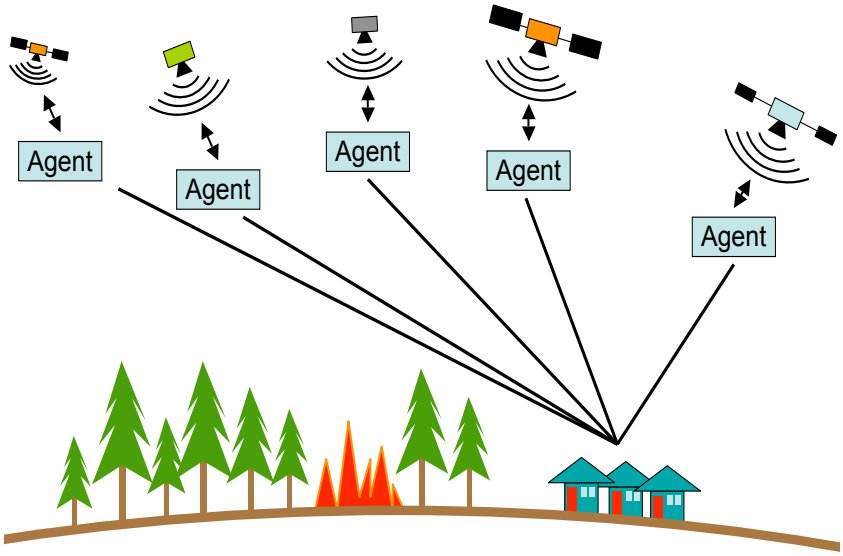

Figure 6.4: Small community needs CubeSat observations of a forest fire. Agents handle observation request. Using one agent per CubeSat, an agent bids for the observation of a particular CubeSat. As a collective agents as a whole must bid for an appropriate number of observations with minimal cost.

consumer makes a request to all of the agents for satellite observations, giving the agents a utility equation representing the value of the benefit it would receive from different numbers and types of observations. Each agent then makes a bid for an observations, using a bidding policy. This policy is evolved from a population of policies, using the value benefit equation given by the consumer, in combination with the agent's model of its CubeSat. These bids take into account the value of an observation, the likelihood that the CubeSat will be able to carry out an observation, and the likelihood that the CubeSat will be willing to carry out an observation given the value of a bid. All these values have uncertainty, making this a difficult problem. In addition the agents will need to coordinate the evolution

of their policies so that the collection of observations derived from all the winning bids is beneficial to the customer and bought at a low cost. In this chapter, we explore these aspects in more detail.

6.1.2.3 Coordinating CubeSats

Academic and industrial programs continue to launch CubeSats equipped with scientific instruments into Low Earth Orbit (LEO) [76]. The capabilities of individual CubeSats are fairly limited due to their size and mass restrictions. Yet, coordinating multiple CubesSats and collecting their combined resources greatly increases their overall value. In this paper, we focus on a particular instance of a CubeSat coordination problem where a customer can coordinate resource purchases for a set of existing CubeSats.

6.1.2.4 Observational Values

Here we assume that there is a set of Earth-observing CubeSats in low Earth orbits, where each satellite is owned by a separate institution [2]. Each of these CubeSats is interested in observations of a particular geographic region of interest for reasons such as crop monitoring, volcano monitoring, fire monitoring, reconnaissance, search and rescue, and weather monitoring. We assume that each CubeSat places some value on observing a particular point of interest (POI), but is able to observe

[2]For simplicity, we will anthropomorphize the CubeSats by treating a CubeSat and the institutions that own it, as the same thing.

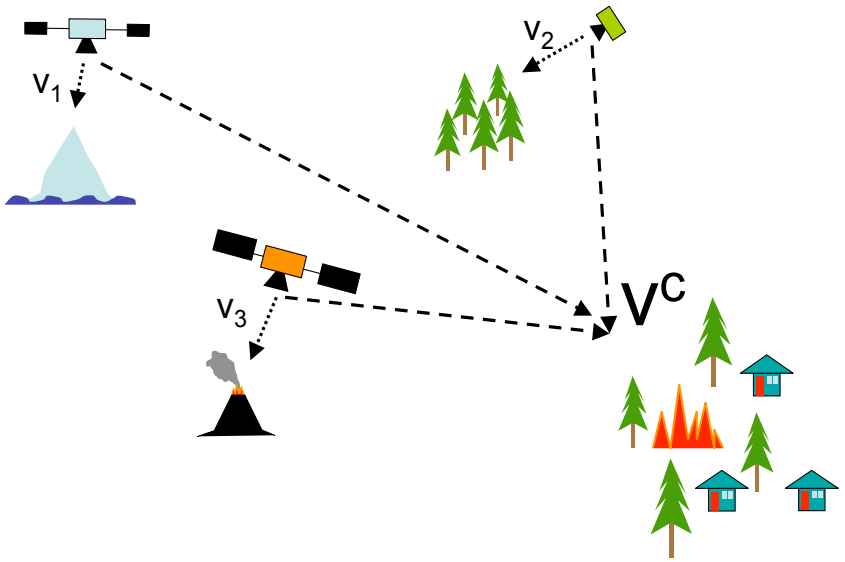

Figure 6.5: A set of CubeSats gain different levels of value, v_i, from observing their own point of interest. A potential customer would like satellites observations of its own point of interest. The value of these observations to the customer, v^c depend on mix and number and locations of satellites involved.

any region of interest beneath its orbit (see Figure 6.5). Each CubeSat places a different value on monitoring its own POI. In addition, this value depends on the distance between the CubeSat and its POI. In general the further the CubeSat is from its POI the less value it will have in monitoring it. Formally we express this value for CubeSat i as:

$$v_i(d_i^p) \ , \tag{6.9}$$

where d_i^p is the distance between the CubeSat and its POI.

The goal of this paper is to figure out how a customer, with no satellites of his own, can make use of these existing satellites to observe a point of interest that this customer is interested in. In general the value of a set of CubeSat observations

to a customer is a function of the observational capabilities of the CubeSats (e.g. resolution, heat sensing, particle sensing, etc.) and the distance of the satellite to the customer's POI. Formally we define this value function for the customer as:

$$v^c(d^c) \ , \tag{6.10}$$

where d^c is the vector of distances between the CubeSats and the customer's POI (note for simplicity the observational capabilities are rolled into v^c). In general, the more observations are better and observations closer to the customer's POI are better.

6.1.2.5 Customer Objective

The objective of the customer is find a set set of observations that have high value to him, v^c, at a low cost:

$$G(d^c, d^p) = v^c(d^c) - \sum_i c_i(d_i^p) \ , \tag{6.11}$$

where c_i is the cost paid to get an observation from CubeSat i. While we assume that there is no direct cost for a CubeSat to observe a POI, we assume that a CubeSat will not make an observation for a customer unless it is paid approximately its opportunity cost for not observing its own POI $v_i(d_i^p)$. Therefore in generally the cost paid $c_i(d_i^p)$ will be higher than opportunity cost $v_i(d_i^p)$ for successful bids. While in some cases this opportunity cost is very high and a CubeSat will never

offer to make an observation for a customer, in other cases in could be close to zero, especially if the CubeSat is not within range of its own POI.

6.1.2.6 Agent Model

Our overall goal is to figure out how a customer can maximize G; i.e. purchase a set of observations that have high value to him at a low cost. In general this will be possible when a set of CubeSats with appropriate capabilities is close to the customer's POI, increasing his value, and far from their own POIs, reducing their opportunity cost. This leads to our central problem: How can a customer sensibly buy a set of satellite resources, when he knows little about the CubeSats' capabilities, their cost model, or even their willingness observe the customer's POI?

We propose to address this problem using agents combined with reinforcement learning (Chapter 2). In this paradigm a single agent is assigned to a single CubeSat, and the action of an agent is to bid on the observations from its CubeSat on behalf of the customer. As a whole, the goal of the agents is to balance the value of all the observations to the customer, v^c, with the cost of these observations. Note that to do this effectively each agent has to take into account both local and global considerations: 1) Locally an agent will make bids when its CubeSat has likely high value, such as when it is close to the customer's POI, and when its CubeSat has likely low cost, such as when the CubeSat is far from its own POI, 2) Globally an agent should only bid for an observation, when that observation increases the total value, v^c, for the customer - agents should not bid for observations that are

not needed or have low marginal value.

For an agent to accomplish its task, we assume that each agent is given the value function for the customer $v^c(d^c)$. We also assume that each agent has some approximate model for the value of its CubeSat, $v_i(d_i^p)$. However, we make little assumptions of the quality of this model or where it came from. For some agents $v_i(d_i^p)$, may be given to it directly from the CubeSat. For other agents, its model for $v_i(d_i^p)$ may be a crude approximation generated through previous interactions with the CubeSat. Given these value functions, we have the agents maximize the customer's objective through evolution.

6.1.3 Batch-CLEAN Rewards for the UAVCN Domain

In this section we calculate the $CB2_{j'}$ Batch-CLEAN rewards for the UAVCN domain by combining Equations 6.7 and 5.2 (These rewards could similarly be derived for the CCD domain, but are excluded here for brevity). Due to the presence of heterogeneous agents in the UAVCN domain, there are two separate types of rewards: one for agents governing transmission power, and one for agents controlling antenna pointing (Section 6.1). $CB2_j$ rewards for power agents are:

$$
\begin{aligned}
&CB2_{j'}(c_{a_{j',k}}) = G(z_T - z_{T,j'} + c_{a_{j',k}}) - G(z_T) = \\
&\frac{B}{m} \sum_{i \notin I_{j'}} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j} S_{i,j}}{F_{i,j'} S_{i,j',c_{a_{j',k}}} + \kappa + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right) \\
&+ \frac{B}{m} \sum_{i \in I_{j'}} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j'} S_{i,j',c_{a_{j',k}}}}{\kappa + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right) \\
&- \frac{B}{m} \sum_{i} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j} S_{i,j}}{\kappa + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right)
\end{aligned}
\tag{6.12}
$$

where $CB2_{j'}$ is an individual reward for the agent controlling transmit power for UAV j', $c_{a_{j',k}}$ is one of the agent's n counterfactual actions, $S_{i,j',c_{a_{j',k}}}$ is the transmission power factor for agent j' calculated with counterfactual action $c_{a_{j',k}}$, and all other variables are as described in Section 6.1. The set of all Batch-CLEAN rewards for agent j' is calculated by calculating $\overline{CB2}_{j'} = \{CB2_{j'}(c_{a_{j',k}})\}_{k=1}^{k \leq n}$, where n is the total number of possible actions the agent can take.

Here, the counterfactual actions only change the power portion and the UAV pointing remains constant. In this setting, these rewards provide the power agents with a reward that tells the effectiveness of its counterfactual transmit power based upon the UAVs current target pointing policy. Thus, during each atomic episode of learning, the agent calculates rewards for each of its potential transmit powers that

approximates the impact each would have on the network bandwidth. Additionally, due to the subtraction between the $G(z_T - z_{T,j'} + c_{a_{j',k}})$ and $G(z_T)$ terms of each reward calculation, much of the impact of the actions of other UAVs is filtered off, resulting in a much clearer learning signal for power agent j'.

Next, we calculated the $CB2_{j'}$ rewards for each UAV's pointing agent, yielding Equation 6.13. Here, $CB2_{j'}$ is an individual reward for the agent controlling the antenna pointing direction for UAV j', $c_{a_{j',k}}$ is one of this pointing agent's n counterfactual actions, $F_{i,j',c_{a_{j',k}}}$ is the signal strength weighting factor based upon the counterfactual pointing action of pointing agent j' for counterfactual $c_{a_{j',k}}$. Again, each agent calculates rewards for each of its potential n counterfactual actions $c_{a_{j',k}}$ during each atomic episode of learning. In this setting, each reward provides the agent with a clear reward signal that tells how each of its potential counterfactual pointing actions $c_{a_{j',k}}$ would impact the network bandwidth.

$$
\begin{aligned}
&CB2_{j'}(c_{a_{j',k}}) = G(z_T - z_{T,j} + c_{a_{j',k}}) - G(z_T) = \\
&\frac{B}{m} \sum_{i \notin I_{j'}} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j} S_{i,j}}{F_{i,j',c_{a_{j',k}}} S_{i,j'} + k + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right) \\
&+ \frac{B}{m} \sum_{i \in I_{j'}} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j',c_{a_{j',k}}} S_{i,j'}}{k + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right) \\
&- \frac{B}{m} \sum_{i} \sum_{j \in J_i} log_2 \left(1 + \frac{F_{i,j} S_{i,j}}{k + \sum_{j \notin J_i} F_{i,j} S_{i,j}} \right)
\end{aligned}
\tag{6.13}
$$

6.2 Results

We conduct experiments in both the UAVCN and CCD domains. We use seven types of agents:

- Agents taking random actions (R)
- Agents learning with Global rewards (G)
- Agents learning with Difference rewards (D)
- Agents learning with CLEAN rewards (C1)
- Agents learning with CLEAN rewards (C2)

- Agents learning with Batch-CLEAN rewards (CB1)
- Agents learning with Batch-CLEAN rewards (CB2)

In all experiments, agents are ϵ-greedy reinforcement learners with a learning rate of $\alpha = 0.20$ and an exploration rate of $\epsilon = 0.10$. All Q-tables were initialized with small random values. All experiments consisted of $r = 30$ statistical runs with $e = 3000$ learning episodes and the error in the mean σ/r was plotted in all experiments.

6.2.1 UAV Communication Network Domain

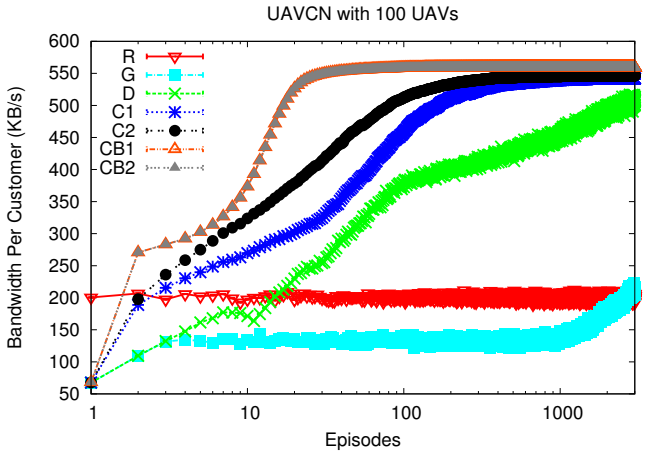

Figure 6.6: 100 UAVs and 100 customers in the UAVCN domain. Agents using CB1 and CB2 rewards learn approximately one-hundred times faster than agents using C1, C2, G and D rewards. Batch-CLEAN rewards outperform D by 20% and G by 80%.

The first experiment involved a set of 100 UAVs and 100 customers within the UAVCN domain. As see in in Figure 6.6, agents taking both random power

and pointing actions perform poorly in this domain. Similarly, agents using global rewards, G, are slow to learn and learn a policy that is far worse than other learning agents. A key reason for this is that global rewards provide a noisy learning signal to agents (each agent's reward depends directly upon the actions of all other agents). Difference rewards, D, address this shortcoming by effectively filtering off much of the impact of other agents on the learning signal. As seen from Figure 6.6, this additional filtering (which addresses structural credit assignment) results increased performance compared to agents using G.

Next, we implemented CLEAN rewards (C1, and C2), which not only address the structural credit assignment problem as difference rewards do, but also address exploratory action noise which is a significant inhibiting factor for multiagent coordination. As seen, CLEAN rewards outperform difference rewards by up to 20% and global rewards by up to 80% in this domain. However, CLEAN rewards still learn slow in comparison to Batch-CLEAN rewards as they are limited by the fact that agents only receive a single reward update per episode of learning.

Batch-CLEAN rewards maintain the performance of CLEAN rewards (as they are based upon the same structure), while at the same time significantly improving learning speed over standard CLEAN rewards. The speed-up in learning stems from the fact that Batch-CLEAN rewards leverage the properties of privatized exploration which were developed to create CLEAN rewards to enable agents to concurrently calculate reward updates for multiple actions during each episode of learning. This extension enables learning speed to be improved over traditional reward techniques such as global or difference rewards, as well as standard CLEAN

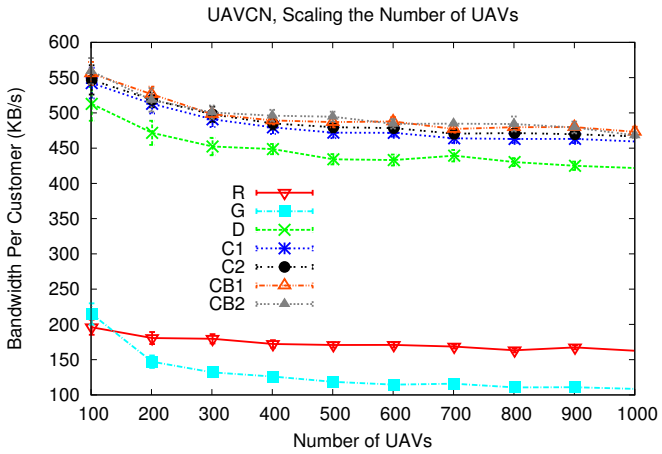

Figure 6.7: Scaling the number of UAVs in the UAVCN domain. Agents using Batch-CLEAN rewards CB1 and CB2 continue to maintain the same converged performance as standard CLEAN rewards C1 and C2 (although they converge more quickly), and outperform the next best method D by at approximately 15-20% when scaling between 100 and 1000 UAVs.

rewards, which only allow a single reward update per learning episode.

As seen from Figure 6.6, Batch-CLEAN rewards converge to a better policy than do agents using both global and difference rewards. The increased performance is due to the structural credit assignment problem and the removal of exploration noise during learning. The improved learning rate is due to the fact that each agent is receiving multiple action updates per episode of learning. It is interesting to note that although the computational increase for agents' reward calculations using CB1 and CB2 rewards compared to G, D, C1, and C2 rewards is ten fold, these agents learn one hundred fold faster (showing a nonlinear increase in learning performance compared to computational costs). Here, Batch-CLEAN rewards CB1 and CB2 outperform agents using D by approximately 20% and agents

using G by 80%, while maintaining the performance of standard CLEAN rewards.

The next set of experiments involved scaling the number of UAVs and customers in the system. The number of UAVs and customers was scaled proportionally in a 1:1 ratio. As seen in Figure 6.7, the general performance of all techniques slightly decreases with scaling. This is to be expected, as the amount of noise present in the system increases as the number of communication towers (UAVs) increases. In a shared-channel network, this background noise necessarily reduces the amount of signal throughput to any individual node in the system. It is important to note however, that the total system bandwidth increases as the number of UAVs is increased, even though the signal per customer gracefully degrades. As seen, agents using random policies, R, continue to perform poorly as scaling increases. Similarly, agents using global rewards, G, experience a 40% drop in performance. Agents using D experience a 20% decrease in performance with scaling up to 1000 agents. Agents using both CLEAN and Batch-CLEAN rewards experience only a 13% drop in performance and the degradation plateaus and stabilizes as scaling increases. The key here is that both CLEAN and Batch-CLEAN rewards outperform all other methods, however, Batch-CLEAN still learns significantly faster than CLEAN in all of these cases (although this cannot be seen by the convergence graph provided).

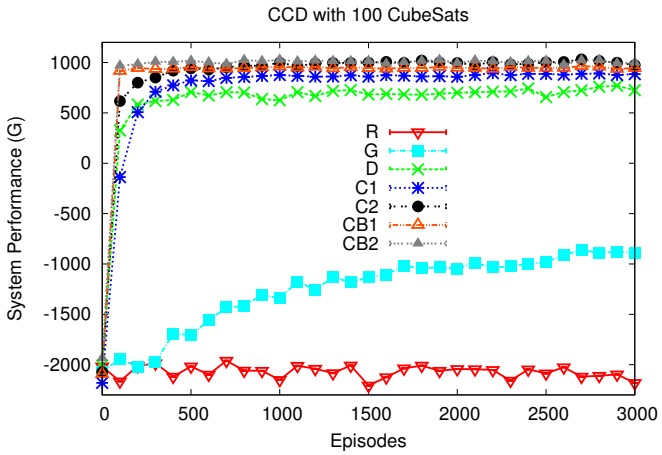

Figure 6.8: 100 agents in the CCD. As seen here, agents using Batch-CLEAN rewards CB1 and CB2 converge significantly faster than other learning methods, including agents using standard CLEAN rewards. Additionally, agents using Batch-CLEAN achieve the same level of performance as standard CLEAN rewards and outperforming D by 30% and G by 600%.

6.2.2 CubeSat Coordination Domain

Figure 6.8 shows the performance of agents within a 100 CubeSat system. As seen, agents following random policies perform poorly in this domain. Similarly, agents using global rewards, G, have difficulty coordinating their actions, leading to poor performance. Agents using difference rewards perform significantly better than these techniques as they attempt to address the structural credit assignment problem present within learning-based systems. However, these rewards alone are still insufficient as they do not address the issue of learning noise associated with exploration which complicates coordination. As seen, CLEAN rewards which address exploratory action noise again produce significantly better performance than both global and difference rewards, however, they still only provide agents

with a single reward update which leads to slow learning.

Batch-CLEAN rewards achieve the same performance as CLEAN rewards as they are based upon the same underlying principles and reward structure, while at the same time significantly improving the learning speed of agents as compared to CLEAN rewards. As seen, Batch-CLEAN techniques learn up to an order of magnitude faster than agents using both difference rewards and standard CLEAN rewards.

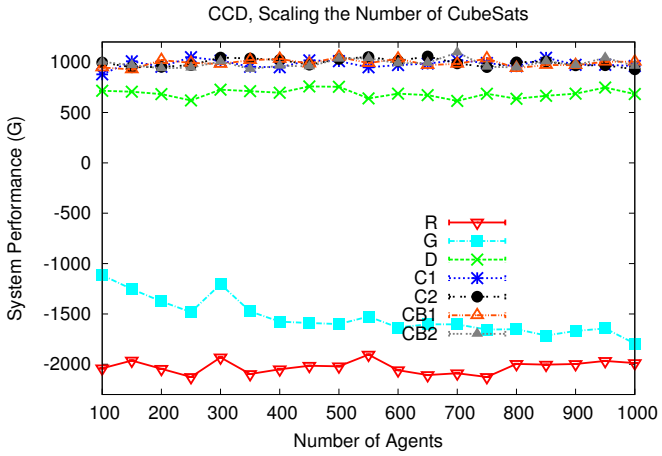

Figure 6.9: Scaling the number of agents in the CCD. As seen, agents using CB1 and CB2 rewards maintain performance as scaling increases and continue to outperform all other methods by at least 30%.

The final set of experiments involves scaling the number of agents within the CCD domain. Again, agents using random rewards perform poorly as they are blindly making bids for satellite observations, without any regard for the ramifications or outcomes from those transactions. Agents using G have a similar pitfall. Although these agents are intelligently making bids and attempting to

learn, agents have difficulty determining the impact of their own bids compared to the bids of other agents within the system. Agents using D are better able to make decisions as each agent's reward is heavily dependent upon its own actions. However, difference rewards do not address exploratory action noise, which results in significantly impeded learning performance, especially as scaling increases and exploration noise becomes an increasingly significant factor. To address this, standard CLEAN rewards were implemented, and are shown to converge to significantly better performance. However, these rewards are still prohibitively slow for many situations due to the fact that they only provide each agent with a single reward update per episode of learning. Batch-CLEAN rewards are used again here to not only achieve the same coordination and performance benefits as standard CLEAN rewards, but to also significantly improve the overall learning speed for agents within the system, enabling agents to be more adaptable to sudden and unplanned changes within the environment.

6.3 Discussion

Existing adaptive learning-based control techniques are relatively inefficient in the way they utilize data (e.g., data from individual interactions with their environment). In particular, existing techniques are only able to compute a single update for their control policy for each piece of data they receive [16, 72, 125]. This frequently results in slow learning, which can be problematic in a number of settings including systems with large numbers of agents where the system size significantly

increases the complexity of the control problem which renders single control updates per piece of data to be prohibitively slow (e.g., the environment may change faster than the agents are able to adapt their policies). Effective multiagent control techniques must be able to make more efficient use of data in order to increase the speed at which they can reconfigure their control policies.

In everyday experience we can only take one mutually exclusive action at a time. For instance we cannot go up *and* down simultaneously. This follows through to traditional learning systems where an agent takes an action and receives a reward based on that action [3]. However, with the CLEAN rewards paradigm, described in the previous chapter, agents take two actions at the same time: 1) They take their public non-exploratory action that is seen by others, and 2) They take their private counterfactual exploratory action. Given the framework needed to enable these counterfactual explorations, we can in fact perform additional exploratory actions at the same time. In this chapter, we proposed batch-CLEAN rewards which enabled agents to make multiple updates during each individual interaction with their environment. This resulted in significantly improved performance and learning speed over existing state-of-the-art techniques (e.g., difference rewards). Such techniques are beneficial to learning in multiagent systems in that it enables the agents to make more efficient use of their data (i.e., interactions with the environment) through leveraging the concepts of "counterfactual action updates" and CLEAN learning (developed in Chapter 3) to enable multiple control updates

[3]To simply discussion, actions discussed here are mutually exclusive so we are not included cases where agents can take multiple non-mutually exclusive actions

per interaction with the environment.

Though these results are encouraging, there are multiple areas for further investigation with regards to rewards based upon counterfactual actions. For example, the ability of such rewards to handle domains involving large numbers of agent actions needs to be explored. The current Batch-CLEAN techniques were applied to domains with up to one hundred joint actions, but these techniques may be prohibitively slow in domains where thousands of actions are present. In such cases, it may be beneficial to provide updates to a particular subset of the potential actions. Additionally, extensions of this work that enable Batch-CLEAN rewards to be computed without the agents having an a priori model of the system objective function would enable them to be utilized within initially unknown environments, which is indicative of most real-world multiagent systems.

Chapter 7 – Utilizing Function Approximation for Learning with Shaped Rewards in Unknown Environments

As we have shown previously, CLEAN reward techniques are capable of addressing the complex coordination problems that arise with decentralized control of large distributed systems such as satellite constellations, UAV swarms, and sensor networks when an accurate model of the system objective function is provided to the agents a priori and the agents can use this model to calculate their CLEAN rewards. Unfortunately, in many real-world systems an accurate model of the system objective function is not readily available to agents, meaning that CLEAN rewards cannot readily be calculated. In this chapter, we address this shortcoming by enabling agents to construct their own approximate model of the system objective by repeatedly interacting with the environment. Here, agents utilize statistical function approximation tools (e.g., neural networks) to construct an approximate model of the system objective and then use this approximate model to calculate their CLEAN rewards. Enabling agents to generate their own approximate model of the system objective which can then be used to calculate their CLEAN rewards extends these techniques into many real-world domains where agents must operate within initially unknown environments, and a priori system knowledge is not readily available.

7.1 Introduction

A key area of research within the area of MARL lies in the design of the rewards agents use to learn. Traditionally, multiagent systems have centered around team reward techniques (i.e., all agents receive the team performance as their individual reward). However, system designers quickly realized that team rewards struggled with scaling to large multiagent systems due to the structural credit assignment problem (i.e., how to assign credit to individual agents for their contribution to the team reward). In order to address this, researchers in the field developed so-called reward shaping techniques (described in Chapter 2 of this work) which provided agents with more individualized reward signals, resulting in improved learning speed and performance.

Potential-based reward shaping was one of the first popular reward shaping technique to arrive on the scene (Chapter 2), where agents would receive the team reward plus an additional shaped reward that was tailored to the specific contributions of each agent. Although potential-based reward shaping techniques showed significant promise in addressing the multiagent coordination problem and provided significant performance gains over traditional team rewards, they had significant information requirements. In particular, potential-based reward shaping techniques require that the system designer has access to a complete system model a priori (i.e., requires the system designer to have both an accurate model of the transition probabilities as well as the system objective). Such requirements are unrealistic in many domains, where this information is simply not available.

In the late 1990's difference rewards, which were a form of shaped rewards based upon the film "its a wonderful life", were introduced. These rewards enabled each agent to calculate the performance of the system when they were present as well as the performance of the system had they not been present. Here, the agents receive a reward that is the difference between these two factors, resulting in a reward that provides each agent with its "net impact" on the system performance. Difference rewards resulted in significantly improved performance and scalability compared to traditional team rewards. Additionally, difference rewards are advantageous over many existing reward shaping techniques as they do not require a complete system model to be computed, and instead require only an accurate partial model of the system (i.e., they only require an accurate model of the system objective to be known).

In this work, we introduced CLEAN rewards which are shaped rewards that are similar to difference rewards in that they provide agents with feedback on their "net impact" on the system and only require an accurate model of the system objective to be computed. However, CLEAN rewards also address issues associated with exploratory action noise (as shown in prior chapters), which leads to significantly improved learning performance compared to difference rewards. In this chapter, we look to further advance the state-of-the-art by enabling agents to use CLEAN rewards to learn within environments where agents have no access to an accurate system model, and instead must construct their own approximate model of the system objective and then use the approximate model to calculate their individual CLEAN rewards.

7.2 Background and Related Work

In cooperative multiagent systems, coordinating the joint actions of agents is difficult [43, 45, 51]. One of the fundamental difficulties in such multiagent systems is the slow learning process where an agent may not only need to learn how to behave in a complex environment, but may also need to account for the actions of the other learning agents [13, 116, 125]. This is especially difficult due to the fact that agents are routinely taking exploratory actions which are observed "publicly" by other agents in the environment. Here, the inability of agents to distinguish the true environmental dynamics from those caused by the stochastic exploratory actions of other agents creates noise on each agent's reward signal [124, 21, 143]. Under these conditions, the solution (learning agents) actually becomes part of the problem. This problem is known as *exploratory action noise* and it can have a significant detrimental impact on learning performance within multiagent systems [61]. CLEAN rewards address this issue by enabling agents to shift from explicit "public" exploration strategies towards implicit "private" exploration strategies. Here, instead of explicitly taking an exploratory action within the environment, each agent utilizes its own local reward model to implicitly approximate the reward it would have received for taking a given exploratory action. In this setting, agents are still able to explore, but their exploratory actions do not impact the learning of other agents within the system. Although as we have shown in previous chapters, CLEAN rewards show significant promise for addressing the coordination problem within multiagent systems, they currently require an accurate partial system

model to be computed. In this chapter, we address this shortcoming by enabling agents to utilize reward modeling techniques to construct their own approximate model of the system objective when it is otherwise unavailable. The agents then use this approximate model to calculate their individual CLEAN rewards during learning.

7.2.1 Neural Networks

A multilayer perceptron (MLP) neural network is a feed-forward supervised learning artificial neural network model. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron with a *nonlinear activation function* (in this work we use a sigmoid activation function). A MLP can be trained with back propagation.

Neural networks (NN) have been used in many applications including: control problems, classification, function approximation, and nonlinear signal-processing [36, 52, 66, 83, 92]. They are useful tools for representing functions and can represent both discontinuous and continuous functions to arbitrary accuracy [17, 36, 40, 52]. Neural networks are biologically inspired mathematical tools modeled after the function of the brain. Each neural network maps a set of inputs to a set of outputs. A neural network consists of a set of input nodes, output nodes, and hidden layer nodes which are connected with weights assigned to each connection (Figure 7.1). The weighted sum of the connections mapped from the inputs through

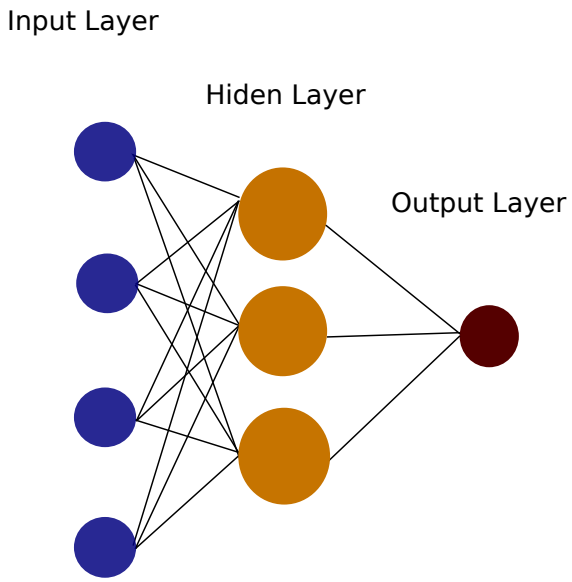

Figure 7.1: Network diagram for a two layer feed forward Neural Network (NN). The input, hidden and output variables are represented by nodes, and the weight parameters are represented by links between the nodes. In feed forward NN the information flow through the network from input to hidden and then to output layer.

the nodes of the neural network generate a set of outputs (where each node itself has an activation function, which is typically nonlinear) [48]. There are many ways of training a neural network including supervised, unsupervised, and reward based methods. A neural network controller utilizing supervised learning has a "teacher" that knows the correct mapping from inputs to outputs, and provides the neural network with constructive feedback on actions taken (improving the neural networks performance over time). Using unsupervised learning with a neural network attempts to use the neural network to cluster a set of unlabeled data using its similarities. Reward based learning with neural networks on the other

hand utilize reward feedback based upon actions taken in order to update the neural networks [17, 40, 52, 83, 92]. In this work, a single hidden layer neural network was used to approximate the underlying system reward function through incrementally interacting with the environment.

In this work, our neural networks utilize a sigmoid activation function (Equation 7.1) ranging from 0 to 1.

$$\phi_1(\mathbf{x}) = \frac{1}{(1 + e^{-\mathbf{w}^T\mathbf{x}})} \tag{7.1}$$

Each layer n of our neural networks consist of q neurons, where $\mathbf{x}_n$ are the activations of the neurons and $\mathbf{y}_n$ are the outputs of the neurons derived by putting the weighted activations through the activation function ϕ [66]. The neural networks used are fully connected, and the output of layer $n-1$ are connected to layer n's inputs $\mathbf{x}_n$ via weights. The weight matrix is defined as $\mathbf{W}_n$ where the elements $w_{i,j}$ corresponds to the weight from output j of the $n-1$ layer to the input i of the n layer [66]:

$$\mathbf{W}_n = \begin{matrix} w_{0,0} & \dots & w_{0,p} \\ \dots & \dots & \dots \\ w_{q,0} & \dots & w_{q,p} \end{matrix} .$$

Therefore $\mathbf{x}_n$ is:

$$\mathbf{x}_n = \mathbf{W}_n\mathbf{y}_{n-1} \tag{7.2}$$

and

$$\mathbf{y}_n = \phi(\mathbf{W}_n \mathbf{y}_{n-1}) \tag{7.3}$$

Where $\mathbf{x}_n$ is the vector of inputs to layer n, W_n is the set of weights on the inputs from layer $n-1$ to layer n, $\mathbf{y}_{n-1}$ is the vector of outputs from layer $n-1$, and $\mathbf{y}_n$ is the vector of neural network outputs at layer n [66]. In this work, we use back propagation to update the weights of the neural network during learning according to the following back propagation update rule:

$$\mathbf{W_n^{t+1}} = \mathbf{W_n^t} + \eta \frac{\partial E}{\partial \mathbf{W}_m} \tag{7.4}$$

Where η is the learning rate parameter. The goal of back propagation is to *backpropagate* our error and apply gradient descent to learn our weights [66]. We refer the interested reader to [25] for additional information on neural networks.

7.3 Reward Shaping within Unknown Environments

In order to calculate shaped rewards which require an accurate system model (e.g., difference rewards or the CLEAN rewards introduced in this section), agents have previously been required to have access to a complete and accurate partial system model (e.g., an analytical model of the team objective $G(z)$). Unfortunately, in many real-world systems the agent-to-agent interactions and system dynamics are too complex to be modeled analytically and are frequently too sophisticated to be

accurately modeled at all, rendering these techniques inadequate for many real-world applications. We address this shortcoming by extending both existing and novel reward shaping techniques (difference rewards and CLEAN rewards, respectively) such that they are able to learn when an accurate analytical model of the team reward is not available. This is achieved by enabling agents to construct their own approximate model of the team objective when the environment is unknown and an accurate model of the team reward is not available a priori. A key benefit of our techniques is that they require the agents to have no a priori system knowledge and they do not require the system designer to make any assumptions about the system, meaning they are completely generalizable to any real-world system.

7.3.1 Reward Modeling

Within some simple multiagent domains, we can expect to be provided with an accurate analytical equation for the team performance objective $G(z)$, which can be used to calculate partial-model based rewards such as difference rewards (Chapter 2). Unfortunately, in many domains (particularly real-world domains) the complex interactions between agents coupled with the system dynamics make the system too sophisticated to be modeled analytically, and in these systems an accurate model of the team reward G is frequently not available. For this sort of "black-box" systems, one simply has the vector z and some reward signal $G(z)$ which tells the team performance.

Our approach to solving this problem is to allow the agents to construct an

approximate model of the team objective $G(z)$ by enabling agents to construct an approximate model of the team objective $f_{G(z)}(z)$ through repeatedly interacting with the environment. In particular, agents will interact with the environment, generating a tuple including the system state vector z and the resultant reward signal $G(z)$, and agents use this information to incrementally train an approximate team reward model. This model can then be used by individual agents to calculate rewards (e.g., difference rewards, CLEAN rewards) which need a model of the team objective in order to learn. For example, with our second variation of CLEAN rewards, the calculation becomes $\mathcal{C}_i = f_{G(z)}(z - z_i + c_i) - f_{G(z)}(z - z_i + c_j)$. In this case, the approximate model of the team objective $f_{G(z)}(z)$ is used in place of the accurate analytical model of the team objective $G(z)$ when calculating both difference rewards and CLEAN rewards. This solution, while conceptually simple, addresses a major criticism of partial model based reward shaping techniques (e.g., difference rewards, CLEAN rewards) in the past: that they are difficult to use if an accurate analytical equation for $G(z)$ is unavailable. In this work we couple these reward modeling techniques with both difference rewards and CLEAN rewards, expanding the applicability of both techniques into a large number of new domains. The key property that distinguishes this approach from standard function approximation is that the structural form of both difference rewards and CLEAN rewards have built-in bias reducers. The subtraction operation ensures systematic errors in the function approximation are eliminated, particularly if the two terms $f_{G(z)}(z)$ and $f_{G(z)}(z - z_i + c_i)$ are close to one another. In this paper, we explore using neural networks (discussed previously) as the function approximators

to model $G(z)$ (although other function approximation techniques such as tabular linear functions and support vector machines are also valid). A key benefit of the techniques used in this work is that we did not bias our reward model in any way and we performed uniformly random sampling of $\{z, G(z)\}$ tuples, meaning that these techniques are completely generalizable to any multiagent system.

7.3.2 CLEAN Rewards for Unknown Environments

Once the agents learn an approximate reward model via repeatedly interacting with their environment and training a function approximator (discussed previously), they will use this reward model to calculate their individual CLEAN rewards.[1]. For example, agents using CLEAN rewards will use the model to calculate their rewards as follows:

$$\mathcal{C}_i = f_{G(z)}(z - z_i + c_i) - f_{G(z)}(z - z_i + c_j) \tag{7.5}$$

where $\mathcal{C}$ is the CLEAN reward of agent η, $G(z)$ is the system objective, $f_{G(z)}(z)$ is the neural network function approximation of the system objective, z is the system state vector containing all key state and action variables. In this work, we assume that the agents have access to the complete z vector, although previous work has shown that similar reward shaping techniques can perform well with only partial z vector information. Here, the precise choice of counterfactual actions c_i and c_j are

[1] Although we also compare the performance of CLEAN rewards with function approximation against difference rewards utilizing the same techniques

dependent upon the specific CLEAN reward design the system designer chooses to use (e.g., Equation 3.2 or 3.3, respectively).

In this setting, agents can begin with no a priori knowledge of their environment, incrementally construct an approximate reward model of the system objective, and then utilize this objective to calculate their shaped CLEAN rewards. In this work, we utilized these techniques in two domains. In this setting, the system is a black box model and agents have no access to the underlying system model other than the team objective function approximation $f_{G(z)}(z)$ they generated.

Chapter 8 – Learning with CLEAN Rewards in Unknown Real-World Domains

Although both CLEAN rewards and Batch-CLEAN rewards have shown significant promise for improving both the speed and performance within multiagent systems, they required an accurate model of the system reward in order to be computed. Unfortunately, in many real-world domains, such information is not available a priori, rending the standard versions of these techniques inadequate. We address the shortcomings of these approaches by enabling agents to learn with both CLEAN and Batch-CLEAN rewards by utilizing model-based reinforcement learning techniques that allow agents to construct their own approximate model of the system reward. Here, agents repeatedly interact with the environment in order to construct an approximate model of the system reward function and then they use this approximate model to compute their individual CLEAN and Batch-CLEAN rewards. In this chapter, we demonstrate this concept for standard CLEAN rewards, although it is a trivial extension to extend these results to Batch-CLEAN rewards.

8.1 Results

In this chapter we utilize two multiagent domains to test the performance of Batch-CLEAN rewards. The first is a UAV Communication Network (UAVCN) domain originally introduced in [7] and previously described in Chapter 6. The second is a CubeSat Coordination Domain (CCD) based upon the idea of satellite resource sharing and fractionated satellite systems which was introduced in [60] and previously outlined in Chapter 6 of this dissertation. We borrowed domain parameter values from those found for these domains within the literature unless otherwise specified [7, 59, 60]. These are the domains outlined in the previous chapter (all necessary domain details including descriptions and equations can be found in the previous chapter). The only key difference between the two implementations is that in this version of the UAVCN domain, agents control power only and always point straight down.

All of these experiments were conducted for 3000 episodes and 30 statistical runs. The learning rate was set to $\alpha = 0.10$ and the exploration rate was set to $\epsilon = 0.05$. The results were validated via a t-test with $t = 0.05$. All agents utilized the standard reinforcement learning update rule: $Q'(s,a) = Q(s,a) + \alpha\left(R(s) - Q(s,a)\right)$. The learning parameters were set to $\alpha = 0.10$ and $\epsilon = 0.10$. In the UAVCN domain, the neural network function approximator had a single input for each agent's action, 100 hidden units (excluding the bias unit), and 1 output unit. In the CubeSats domain, the neural network had a single input for each agent's action, 500 hidden units (excluding the bias unit), and 1 output

unit. In both cases, the neural network was trained on a sample of 100,000,000 data points from the environment (corresponding to only a minute portion of the overall action space which is of size 10^{100}) and were trained using standard back propagation with a neural network learning rate of $\gamma = 0.05$. The neural network weights were initialized randomly between 0 and 0.50.

It is important to note that in these settings, although the neural network function approximator is computationally expensive to train, the overall portion of the joint-action space being explored by the agents to construct this model is miniscule. For example, in the two domains in this work, there are 10^{100} possible joint-actions, and the agents are only exploring one hundred billion of these possibilities. Additionally, it is important to note that there are likely significantly better function approximation techniques that could both improve performance and significantly reduce the number of samples required to train. Here, we chose to use neural network function approximation techniques due to their generality and robustness. Neural networks are a high variance function approximation tool, and it is likely that alternate function approximation techniques (e.g., support vector machines) would be able to perform as well or better than the techniques used here, at a fraction of the computational costs. Additionally, we did not attempt to re-use any of the training data. However, it has been repeatedly shown that techniques such as cross-validation can significantly improve the efficiency of the usage of training data. Combining tools such as cross validation (which attempt to make optimal use of a given set of training data) with alternate function approximation techniques such as support vector machines (which are less computationally ex-

pensive than neural network techniques) would likely result in a reduction in the computational costs of such function approximation techniques. Based upon our experience with in this work, we would predict that a reduction of at least two or three orders of magnitude with regards to the computational cost of training the function approximator can be achieved using these techniques.

8.1.1 UAVCN Domain

First, we applied CLEAN learning with a function approximation of the system model to a state-less action-value domain. Here, we tested the performance of CLEAN rewards using function approximation in a 100 UAV communication network. Initially, the agents began with no knowledge of the system. Then, they incrementally constructed an approximate model of the system's reward function by repeatedly interacting with the environment. This model was then passed out to all the agents and used by each agent to calculate their individual local rewards (e.g., global, difference, and CLEAN rewards, respectively) except in the case of standard global rewards G. Agents using G were provided with a complete and accurate model of the system reward function, as this was to be used for a benchmark to test the performance of the other techniques.

As seen in Figure 8.1, agents using traditional global rewards struggle to coordinate their actions due to their inability to address the structural credit assignment problem. Additionally, agents using global rewards with function approximation, G_{NN} also struggled as they were unable to address the credit assignment problem

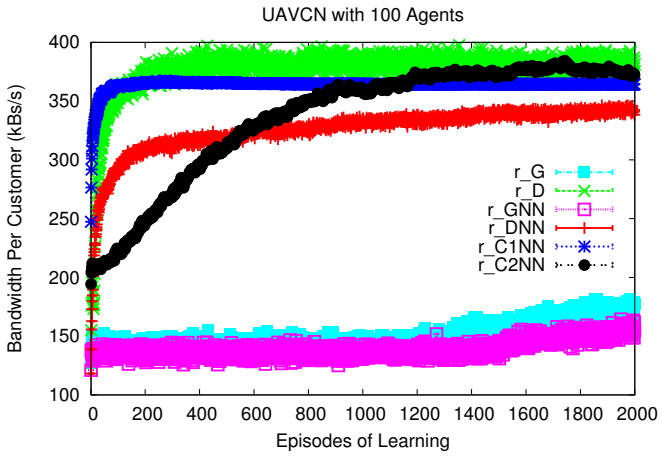

Figure 8.1: 100 agents in the UAVCN domain. As seen, agents using CLEAN rewards with an approximation of the system reward outperform agents using global rewards, G, which have a complete and accurate model of the system reward. Additionally, agents using CLEAN rewards utilizing function approximation (e.g., $C1_{NN}$ and $C2_{NN}$) are able to perform approximately as well as agents using standard difference rewards D with a complete analytical model of the system in this case.

and had an inherently inaccurate system reward model. Agents using difference rewards with function approximation, D_{NN} were able to address these shortcomings in order to slightly outperform traditional global rewards with respect to overall performance (although they learned much slower than agents using traditional global rewards due to the inaccuracies in their reward model as well as their inability to address the coordination issues brought on by exploratory action noise). CLEAN rewards are able to affectively address the credit assignment problem, as well as exploratory action noise in order to achieve significantly improved performance over all other techniques. The key result here is that CLEAN rewards with function approximation techniques are able to significantly outperform global

rewards with a model, and difference rewards with an approximate model. This places CLEAN rewards not only at the forefront of the state-of-the-art with regards to reward shaping techniques that require a model, but at the forefront with respect to shaped rewards that are capable of learning within unknown environments.

8.1.2 CubeSat Coordination Domain

In this setting, we analyze the performance of agents using CLEAN rewards with function approximation in a domain that has both states and actions. Here, we explored the performance of a 100 agents CubeSat system in which the agents initially had no underlying knowledge of their environment. First, the agents learned an approximate model of the system objective of their environment, then this reward model was used by the agents to calculate their individual rewards.

As seen in Figure 8.2, agents using traditional global rewards, G, perform poorly overall. This is because the agents are unable to address the structural credit assignment problem (i.e., how to determine the contribution each individual agent had on the resultant system performance), resulting in poor coordination during learning. Similarly, agents learning using a neural network function approximation of the system objective G_{NN} also perform poorly for the same reasons. It is interesting to note that the system objective model of G achieved very comparable performance to agents using the exact analytical model of the system objective, G. This suggests that accurate models of the underlying system objectives can be learned by agents. Although in the future, less computationally expensive models

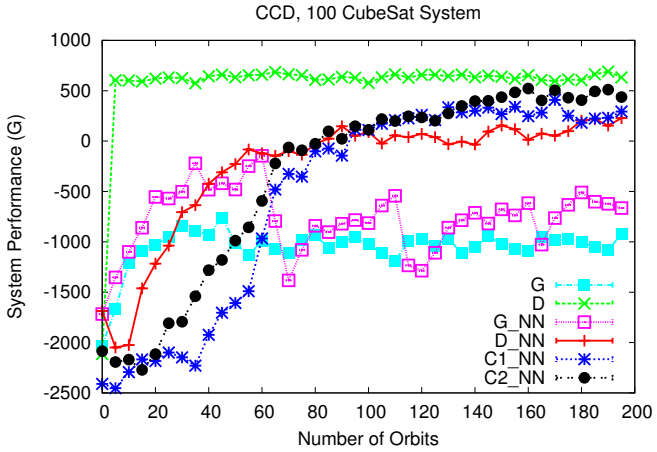

Figure 8.2: 100 agents in the Cubesat Coordination Domain. Agents using traditional global rewards, G, with a complete and accurate analytical model of the system reward function are unable to perform as well as agents using shaped rewards with an approximate system model. Difference rewards utilizing an approximate model of the system reward are able to outperform traditional global rewards due to their ability to improve coordination by addressing the structural credit assignment problem. Agents using CLEAN rewards with an approximate model of the system objective are able to perform nearly as well as difference rewards with an accurate model (D) because they simultaneously address the structural credit assignment problem as well as exploratory action noise.

would be preferred. There are a plethora of function approximation techniques that would likely require significantly less computation burden than neural networks. We chose neural networks because they are effective function approximators in a wide range of situations (it has been said that neural networks are second best at most things).

Figure 8.2 also shows us that agents using difference rewards with function approximation, D_{NN}, outperform agents using both G and G_{NN}, which is noteworthy. As seen here, agents using D_{NN} are able to perform significantly better than agents

using standard global rewards (even when agents using the global rewards have access to a complete and accurate system reward model, G). Here, difference rewards perform better because even with an approximate reward model, their ability to address the structural credit assignment problem enables them to achieve significantly better coordination than global reward techniques. This is an important result because there is currently no work throughout the literature that supports the ability of difference rewards to function effectively when no accurate model of the system is available (especially in a setting like this work, where we use a completely general function approximation technique and we make no assumptions about the system or its behavior when training the function approximator).

Finally, we applied CLEAN rewards with function approximation to this domain, as seen, the CLEAN rewards techniques significantly outperform all of the other methods used in this work. This is because CLEAN rewards are able to address the structural credit assignment problem like difference rewards do, but they are also able to address the issues associated with exploratory action noise. This is an important result for two key reason: i) it demonstrates the ability of CLEAN rewards to learn when an accurate system reward model is unavailable (as is the case in many real-world systems), and ii) it demonstrates that exploratory action noise is still a critically important learning factor, even when the reward model being used to learn is inherently inaccurate.

8.2 Conclusions

Traditional single-agent learning systems have an elegant and intuitive relationship with how we expect learning to work in everyday life: We take an action; That action has consequences; We learn from those consequences. In a multiagent system, this paradigm becomes a little more muddled: Everyone takes actions; these actions have multiple consequences; We all learn from all these consequences. The issue is that the reward we learn from is now a function of actions of other agents and to make things worse, many of these actions are exploratory. The issue of having other agents' actions affecting a reward function can be handled through previously researched counterfactual reward shaping, and the exploratory action noise can be handled through our counterfactual exploration method utilized by CLEAN rewards introduced in Chapter 3. However, both these methods break our elegant learning paradigm: We cannot naturally see the consequences of counterfactual actions on our environment, since these actions never actually happened. Instead we depend on models of how actions lead to their consequences. The difficulty of creating such models range from implementing a few equations to Herculean efforts of combining multiple complex simulators. Making such models efficient leads to even more complexity, increasing the burden to the system designer (who likely wanted to use a learning system to reduce the design burden). In this chapter, we reduced these burdens by enabling agents to build statistical models of their rewards through iteratively interacting with the environment. These statistical models were then utilized in conjunction with CLEAN rewards. The ability to

develop statistical models of the agents' rewards will enable these techniques to be applied to real-world systems.

Although these results were promising, there are several directions for future work. First, the neural network function approximation techniques were not optimized with respect to the neural network parameters (e.g., number of layers, number of nodes, or learning rate) which may have been partly to blame for the significant computational cost of constructing the approximation of the reward model. Second, the neural networks in this work were computationally expensive, in the future alternate function approximation techniques should be used which are able to establish a relatively accurate system model which provides good performance at a fraction of the computational costs of the neural network techniques used in this work. Finally, the function approximation techniques developed in this work were constructed offline and then used online during learning. These techniques must be extended to the case where the approximate reward model can be learned online.

Chapter 9 – Conclusions

Our previous work on multiagent coordination focused on deriving reward functions for agents to ascertain their contribution to a system wide objective [10, 12, 16, 18, 15, 137, 134, 136, 132, 138, 140]. In this work, we used and extended that framework to estimate the impact of agent actions on their own and by extension the system objective functions. In this work, we:

1. Developed a multiagent learning algorithm that accounts for the "background noise" caused by the exploratory actions of all agents (CLEAN Rewards).

2. Developed a "one-action, multiple updates" algorithms to improve scaling and learning speed in multiagent learning (Batch-CLEAN Rewards).

3. Derived statistical models to extend CLEAN rewards to domains where the functional form of the system objective function is unknown.

The long-term goal of this work is to enable the widespread use of multiagent learning in the control and coordination of large systems. To that end, we introduced and investigated a new learning paradigm that explicitly considers the impact of an agent's exploratory "noise" (so-called exploratory action noise) on another agent in a large model-based multiagent reinforcement learning systems. Next, we introduced a novel reward shaping technique called Coordinated Learning without

Exploratory Action Noise which eliminated this noise for agents learning within a multiagent system, resulting in improved learning and performance. Additionally, CLEAN rewards promoted coordination by addressing the structural credit assignment problem present during learning. We then generalized this concept to allow agents to explore and assess the potential values of multiple action for each of their interactions with the environment. Then, we improved the real-world applicability of the developed techniques by reducing the reward model requirements.

The discovery of exploratory action noise and CLEAN rewards is a major contribution to the field of multiagent systems. Developing multiagent learning techniques which account for learning noise caused by the exploratory actions of agents required new theoretical insights (e.g., how to quantify the impact of learning noise on resultant system performance) and practical advances (e.g., how to maintain the exploration necessary for learning, while eliminating the noise it causes in the learning process). This work quantified the impact of exploratory actions on multiagent learning performance and subsequently developed learning techniques (i.e., CLEAN rewards) which eliminated the learning noise associated with such exploration actions. Here, we demonstrated the benefits of CLEAN rewards for learning within multiagent environments which are inherently non-stationary.

Next, we developed Batch-CLEAN which extended the benefits of CLEAN rewards while increasing the learning speed. Eliminating the non-stationarity in the environment for a multiagent systems is a key step in improving system performance, but does not, by itself, rectify the slow convergence speeds of multiagent systems. Indeed, that issue stems also from the exponentially large joint action

spaces prevalent in multiagent systems. The third objective aims at significantly speeding up learning in multiagent systems by enabling agents to perform reward updates for multiple actions after each interaction with the environment. The impact of this objective is to enable agents to search through a much larger portion of the search space, significantly improving the learning speed compared to existing techniques. This is particularly critical to applying multiagent algorithms in real world domains where the dynamic and stochastic environment requires control techniques that can quickly adapt to changing system needs. To address this, Batch-CLEAN rewards enabled agents to calculate reward updates for multiple actions during each time step. This approach maintained the benefits of CLEAN rewards (i.e., performance gains) while at the same time significantly increasing learning speed.

Finally, we extended the real-world applicability of CLEAN rewards based techniques by extending them to domains where the functional form of the system objective is not known or cannot be defined. In many real world systems the environment cannot be queried for the impact of alternative actions without actually taking those actions. Here, we enabled agents to construct statistical models of the system reward function. The agents then used this approximate model of the system reward when calculating their individual CLEAN rewards. Such approximation techniques are ideal for extensions to Batch-CLEAN rewards due to their low computational expense (compared to complete re-simulation).

Although this work significantly advanced the state-of-the-art in multiagent learning by introducing several new reward shaping techniques, there is still a sig-

nificant amount of work to be done. First, a thorough exploration of the types of approximate rewards modeling tools should be conducted, and the trade-offs should be analyzed. It is likely that both reduced computational costs and significant performance gains could be achieved over the simple neural networks applied in this work. In particular, combining cross-validation training with lower variance regression modeling techniques for developing approximate reward models can likely be applied in order to improve performance over the model-based reinforcement learning techniques utilized in this work. Additionally, in order to further improve the real-world applicability of CLEAN rewards, work needs to be done to extend them to domains where agents have neither an accurate system model (i.e., agents use function approximation techniques to develop an approximate system reward model for learning) nor access to information about all other agents in the system at any given time (i.e., under communication restrictions). This work would make these techniques directly applicable to almost any real-world system (i.e., limited information and only an approximate reward model) and these extensions should be fairly straight forward to implement.

Bibliography

[1] O. Abdelkhalik and D. Mortari. Satellite constellation design for earth observation. *15TH AAS/AIAA Space Flight Mechanics Meeting*, 2005.

[2] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Optimal tuning of continual, online, exploration in reinforcement learning. *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 2006.

[3] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Tuning continual exploration in reinforcement learning: An optimal property of the Boltzmann strategy. *Neurocomputing*, 71, 2008.

[4] N. Agmon, C. Fok, Y. Emaliah, P. Stone, C. Julien, and S. Vishwanath. On coordination in practical multi-robot patrol. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2012.

[5] N. Agmon and P. Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[6] N. Agmon, D. Urieli, and P. Stone. Multiagent patrol generalized to complex environmental conditions. *Proceedings of the Twenthy-Fifth Conference on Artificial Intelligence (AAAI)*, 2011.

[7] A. Agogino, C. HolmesParker, and K. Tumer. Evolving large scale uav communication system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Philadelphia, PA, July 2012.

[8] A. Agogino and K. Tumer. Unifying temporal and structural credit assignment problems. *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

[9] A. Agogino and K. Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, NY, July 2004.

[10] A. Agogino and K. Tumer. Multi agent reward analysis for learning in noisy domains. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Utrecht, Netherlands, July 2005.

[11] A. Agogino and K. Tumer. Entropy based anomaly detection applied to space shuttle main engines. In *In Proceedings of the IEEE Aerospace Conference*, 2006.

[12] A. Agogino and K. Tumer. Quicr-learning for multi-agent coordination. *American Association for Artificial Intelligence (AAAI)*, 2006.

[13] A. Agogino and K. Tumer. Analyzing and visualizing multi-agent rewards in dynamic and stochastic domains. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, pages 320–338, 2008.

[14] A. Agogino and K. Tumer. Learning indirect actions in complex domains action suggestions for air traffic control. *Advances in Complex Systems*, 12:493–512, 2009.

[15] A. Agogino and K. Tumer. A multiagent approach to managing air traffic flow. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 2012.

[16] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi Agent Systems*, 17(2):320–338, 2008.

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

[18] A. K. Agogino and K. Tumer. Learning indirect actions in complex domains: Action suggestions for air traffic control. *Advances in Complex Systems*, 12:493–512, 2009.

[19] J. Asmuth, L. Li, M. Littman, A. Nouri, and D. Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of The 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

[20] J. Asmuth, M. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2008.

[21] J. Audibert, R. Munos, and C. Szepesvari. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 2009.

[22] D. Baker and S. Worden. The large benefits of small-satellite missions. *EOS, Transactions American Geophysical Union*, 89(33), 2008.

[23] C. Baldassano and N. Leonard. Explore vs. exploit: Task allocation for multi-robot foraging. *Preprint*, 2012.

[24] S. Barrett and P. Stone. An analysis framework for ad hoc teamwork tasks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

[25] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[26] Y. Bjornsson, V. Hafsteinsson, A. Johannsson, and Einar Jonsson. Efficient use of reinforcement learning in a computer game. *Computer Games - Artificial Intelligence, Design, and Education (CGAIDE)*, 2004.

[27] M. Bowling and M. Veloso. Simultaneous adversarial multi-robot learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[28] Michael Bowling. Convergence and no-regret in multiagent learning. *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[29] O. Buffet, A. Dutech, and F. Charpillet. Shaping multiagent systems with gradient reinforcement learning. *Journal of Autonomous Agents and Multi Agent Systems (JAAMAS)*, 2007.

[30] L. Busoniu, R. Babuska, and B. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C - Applications and Reviews*, 2008.

[31] M. Castronovo, F. Maes, R. Fonteneau, and D. Ernst. Learning exploration/exploitation strategies for single trajectory reinforcement learning. In *Proceedings of the 10th European Workshop on Reinforcement Learning*, 2012.

[32] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning - a bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2003.

[33] G. Chalkiadakis and C. Boutilier. Sequentially optimal repeated coalition formation under uncertainty. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 2012.

[34] D. Challet and N. Johnson. Optimal combination of imperfect objects. *Physics Review Letters 89*, 2002.

[35] Y. Chang, T. Ho, and L. Kaelbling. All learning is local: Multi-agent learning in global reward games. *Computer Science Journal*, 2004.

[36] Tianping Chen and Robert Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, 1995.

[37] M. Chhabra and S. Das. Learning the demand curve in posted-proce digital goods auctions. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[38] C. Claus and C. Boutilier. The dynamics of reinforcement learning cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[39] M. Colby, C. HolmesParker, and K. Tumer. Coordination and control of large distributed sensor networks. In *Future of Instrumentation International Workshop (FIIW)*, 2012.

[40] Mitch Colby, Ehsan Nasroullahi, and Kagan Tumer. Optimizing ballast design of wave energy converters using evolutionary algorithms. July 2011.

[41] V. Conitzer and T. Sandholm. Bl-wolf - a framework for loss-bounded learnability in zero-sum games. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, 2003.

[42] S. Devlin and D. Kudenko. Plan-based reward shaping for multiagent reinforcement learning. *Fourth IEEE International Conference on Intelligent Systems*, 2008.

[43] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. *In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[44] E. Durfee. *Distributed Problem Solving and Planning*. Springer-Verlag New York, Inc, 2001.

[45] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 2005.

[46] A. Farinelli, A. Rogers, and N.R. Jennings. Maximising sensor network efficiency through agent-based coordination of sense/sleep schedules. *In Proceedings of the International Workshop on Energy in Wireless Sensor Networks*, 2008.

[47] F. Fave, A. Rogers, Z. Xu, S. Sukkarieh, and N. Jennings. Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. *IEEE International Conference on Robotics and Automation*, 2012.

[48] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning, 2008.

[49] T. Gabel and M. Riedmiller. Evaluation of batch-mode reinforcement learning methods for solving dec-mdps with changing action sets. In *Recent Advances in Reinforcement Learning - Lecture Notes in Computer Science Volume 5323*, 2008.

[50] A. Galstyan. Continuous strategy replicator dynamics for multi-agent learning. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 2011.

[51] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

[52] Uli Grasemann, Daniel Stronger, and Peter Stone. A neural network-based approach to robot motion control. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup-2007: Robot Soccer World Cup*

XI, volume 5001 of *Lecture Notes in Artificial Intelligence*, pages 480–87. Springer Verlag, Berlin, 2008.

[53] M. Grzes and D. Kudenko. Online learning of shaping rewards in reinforcement learning. *Neural Networks*, 23, 2010.

[54] A. Guez, R. Vincent, M. Avoli, and J. Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2008.

[55] K. Hadeli, P. Valckenaers, C. Zamfirescu, H. Brussel, B. Germain, T. Holvoet, and E. Steegmans. Self-organizing in multiagent coordination and control using stigmergy. In *Engineering Self-Organizing Systems*, 2003.

[56] M. Hapgood, S. Eckersley, R. Lundin, M. Kluge, and U. Prechtel an P. Hyvonen. Nano satellite beacons for space weather monitoring. In *In Proceedings of the 5th ESA Round Table on Micro/Nano Technologies for Space*, 2005.

[57] S. Herwitz, L. Johnson, J. Arvesen, R. Higgins, J. Leung, and S. Dunagan. Precision agriculture as a commercial application for solar-powered unmanned aerial vehicles. In *American Institute of Aeronautics and Astronautics (AIAA)*, 2002.

[58] T. Hester and P. Stone. Learning and using models. *Reinforcement Learning: State of the Art*, 2011.

[59] C. HolmesParker and A. Agogino. Agent-based resource allocation in dynamic cubesat constellations (extended abstract). In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[60] C. HolmesParker, A. Agogino, and K. Tumer. Evolving distributed resource sharing for cubesat constellations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Philadelphia, PA, July 2012.

[61] C. HolmesParker, A. Agogino, and K. Tumer. Clean rewards for improving multiagent coordination in the presence of exploration (extended abstract). In *the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[62] C. HolmesParker, A. Agogino, and K. Tumer. Exploiting structure and utilizing agent-centric rewards to promote coordination in large multiagent systems (extended abstract). In *the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[63] C. HolmesParker and K. Tumer. Combining difference rewards and hierarchies for scaling to large multiagent systems. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA)*, 2012.

[64] J. Hu and M. Wellman. Multiagent reinforcement learning - theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML)*, 1998.

[65] J. Hu and M. Wellman. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the Second International Conference on Autonomous Agents*, 1998.

[66] J.L. Hudson, M. Kube, R.A. Adomaitis, I.G. Kevrekidis, A. Lapedes, and R. Farbar. Nonlinear signal processing using neural networks: Prediction and system modeling. In *Los Alamos National Laboratory Theoretical Division*, pages 2075–2081, July 1987.

[67] A. Iscen, C. HolmesParker, and K. Tumer. Handling communication restrictions with shaped rewards. In *Adaptive and Learning Agents Workshop (ALA 2011)*, 2011.

[68] M. Jain, M. Taylor, and M. Tambe. Dcops meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. *International Joint Conference on Artificial Intelligence*, 2009.

[69] N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 1998.

[70] S. Jensen, D. Boley, M. Gini, and P. Schrater. Non-stationary policy learning in 2-player zero sum games. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2005.

[71] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning - a survey. *Journal of Artificial Intelligence Research (JAIR)*, 1996.

[72] S. Kalyanakrishnan and P. Stone. Batch reinforcement learning in a complex domain. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007.

[73] S. Kamboj, W. Kempton, and K. Decker. Deploying power grid-integrated electric vehicles as a multiagent system. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[74] S. Kar, J. Moura, and H. Poor. Qd-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus and innovations. *IEEE Transactions on Signal Processing*, 2012.

[75] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

[76] B. Klofas, J. Anderson, and K. Leveque. A survey of cubesat communication systems. Technical report, California Polytechnic State University, 2008.

[77] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 1997.

[78] P. Krysta, T. Michalak, T. Sandholm, and M. Wooldridge. Combinatorial auctions with externalities. *9th International Conference on Autonomous Agents and Multiagent Systems*, May 2010.

[79] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

[80] L. Li, M. Littman, and C. Mansley. Online exploration in least-squares policy iteration. *Proceedings of the Eight International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

[81] Z. Li, J. Chen, and E. Baltsavias, editors. *Advances in Photogrammetry, Remote Sensing, and Spatial Information Sciences: 2008 ISPRS Congress Book*, London, United Kingdom, 2008. The Taylor Francis Group.

[82] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 1992.

[83] Peter X. Liu, Ming J. Zuo, and Max Q.-H. Meng. Using neural network function approximation for optimal design of continuous-state parallel-series systems. *Computers & Operations research*, 30(339-352), July 2001.

[84] M. Lopes, T. Lang, M. Toussaint, and P. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[85] K. Macarthur, R. Stranders, S. Ramchurn, and N. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

[86] H. Maei, C. Szepesvari, S. Bhatnagar, and R. Sutton. Toward off-policy learning control with function approximation. In *In Proceedings of the 27th International Conference on Machine Learning*, 2010.

[87] S. McArthur, E. Davidson, V. Catterson, A. Dimeas, N. Hatziagyriou, F. Ponci, and T. Funabashi. Multi-agent systems for power engineering applications - part i: Concepts, approaches, and technical challenges. *IEEE Transactions on Power Systems*, 22(4), 2007.

[88] S. McArthur, E. Davidson, V. Catterson, A. Dimeas, N. Hatziagyriou, F. Ponci, and T. Funabashi. Multi-agent systems for power engineering applications - part ii: Technologies, standards, and tools for building multi-agent systems. *IEEE Transactions on Power Systems*, 22(4), 2007.

[89] N. Monekosso, P. Remagnino, and A. Szarowicz. An improved q-learning algorithm using synthetic pheromones. *Lecture Notes in Computer Science*, 2296, 2001.

[90] A. Muteanu. Nanosat/cubesat constellation concepts. Masters thesis, Cranfield University, Bedfordshire, UK, 2009. Thesis.

[91] R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems*, 2003.

[92] K. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, March 1990.

[93] K. Narendra and M. Thathachar. *Learning Automata - An Introduction*. Prentice Hall, 1989.

[94] E. Nasroullahi and K. Tumer. Combining coordination mechanisms to improve the performance of multi-robot teams. In *Artificial Intelligence Research*, 2012.

[95] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[96] T. Nguyen, T. Silander, and T. Leong. Transferring expectations in model-based reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[97] A. Nouri and M. Littman. Multi-resoultion exploration in continuous spaces. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2009.

[98] A. Nouri and M. Littman. Dimension reduction and its application to exploration in continuous state spaces. *Machine Learning Journal*, 2010.

[99] O. Ozcan and J. Alt. Balancing exploration and exploitation ratio in reinforcement learning. In *Proceedings of the 2011 Military Modeling and Simulation Symposium*, 2011.

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

[101] N. Picon. Effects of wireless power beaming in the space industry: Modern applications and future possibilities. *Triple Helix - Explorations in Science, Society and Law*, 2011.

[102] P. Poupart and N. Vlassis. Model-based bayesian reinforcement learning in partially observable domains. *The International Symposium on Artificial Intelligence and Mathematics*, 2008.

[103] S. Proper and K. Tumer. Modeling difference rewards for multiagent learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

[104] QinetiQ. http://www.qinetiq.com/pages/default.aspx. *Online*, 2011.

[105] S. Ramchurn, P. Vytelingum, A. Rogers, and N. Jennings. Agent-based control for the decentralised demand side management in the smart grid. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[106] D. Ray, A. Mandal, S. Mazumder, and S. Mukhopadhay. Application of single agent q-learning for light exploration. *IEEE*, 2010.

[107] S. Ray and P. Tadepalli. Model-based reinforcement learning. *Encyclopedia of Machine Learning*, 2010.

[108] L. Rejeb, Z. Guessoum, and R. MHallah. The exploration-exploitation dilemma for adaptive agents. In *Proceedings of the Fifth European Workshop on Adaptive Agents and Multi-Agent Systems*, 2005.

[109] S. Reynolds. Reinforcement learning with exploration. *PhD Dissertation, University of Birmingham, United Kingdom*, 2002.

[110] V. Robu, I. Vetsikas, E. Gerding, and N. Jennings. Flexibly priced options a new mechanism for sequential auction markets with complementary goods (extended abstract). *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 1485–1486, May 2010.

[111] A. Rogers, A. Farinelli, and N. Jennings. Self-organising sensors for wide area surveillance using the max-sum algorithm. *Lecture Notes in Computer Science. Self-Organizing Architectures*, 2010.

[112] A.Q. Rogers and L.J. Paxton. Small satellite constellations for space weather and space environment measurements. *International Astronautical Congress*, 2008.

[113] G. Romeo, G. Frulla, and E. Cestino. Design of solar high altitude long endurance aircraft for multi payload and operations. *Aerospace Science and Technology*, 10(6):541 – 550, 2006.

[114] S. Ross and J. Pineau. Model-based bayesian reinforcement learning in large structured domains. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[115] H. Runge, W. Rack, A. Ruiz-Leon, and M. Hepperle. A solar powered hale-uav for arctic research. *1st CEAS European Air and Space Conference*, 2007.

[116] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach (Third Edition)*. Prentice Hall, Pearson Publication Inc, 2010.

[117] R. Sandau, H. Roser, and A. Valenzuala, editors. *Small Satellite Missions for Earth Observations New Developments and Trends*, New York, NY, 2010. Springer Heidelburg Dordrecht London.

[118] K. Schilling. Networked distributed pico-satellite systems for earth observation and telecommunication applications. Technical report, Julius-Maximilians Universitat Wurzburg, 2008.

[119] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 2006.

[120] A. Simpkins, R. Callafon, and E. Todorov. Optimal trade-off between exploration and exploitation. *American Control Conference*, 2008.

[121] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Online adaptation of game opponent ai in simulation and in practice. In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON)*, 2003.

[122] P. Stone. *Layered Learning in Multiagent Systems - A Winning Approach to Robotic Soccer*. MIT Press, 2000.

[123] P. Stone and M. Veloso. Multiagent systems - a survey from a machine learning perspective. *Autonomous Robots*, 2000.

[124] A. Strehl. Probably approximately correct (pac) exploration in reinforcement learning. In *Dissertation*, 2007.

[125] R. Sutton and A. Barto. *Reinforcement Learning An Introduction*. MIT Press, Cambridge, MA, 1998.

[126] M. Tambe. Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, 1997.

[127] M. Taylor, M. Jain, P. Tandon, and M. Tambe. Using dcops to balance exploration and exploitation in time-critical domains. In *Proceedings of the Annual Workshop on Distributed Constraint Reasoning at IJCAI-09*, 2009.

[128] M. Taylor, M. Jain, P. Tandon, M. Yokoo, and M. Tambe. Distributed online multi-agent optimization under uncertainty balancing exploration and exploitation. *Advances in Complex Systems*, 2011.

[129] W. Teacy, G. Chalkiadakis, A. Farinelli, A. Rogers, N. Jennings, S. McClean, and G. Parr. Decentralized bayesian reinforcement learning for online agent collaboration. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

[130] S. Thrun. The role of exploration in learning control. *Handbook for Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, 1992.

[131] S. Thrun. *Exploration in Active Learning.* Handbook of Brain and Cognitive Science, MIT Press, 1995.

[132] K. Tumer. Designing agent utilities for coordinated, scalable and robust multi-agent systems. In P. Scerri, R. Mailler, and R. Vincent, editors, *Challenges in the Coordination of Large Scale Multiagent Systems*, pages 173–178. Springer, 2005.

[133] K. Tumer and A. Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[134] K. Tumer and A. Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[135] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 330–337, Honolulu, HI, May 2007.

[136] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 330–337, Honolulu, HI, May 2007.

[137] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International*

Joint Conference on Autonomous Agents and Multi-Agent Systems, pages 378–385, Bologna, Italy, July 2002.

[138] K. Tumer, A. K. Agogino, and Z. Welch. Traffic congestion management as a learning agent coordination problem. In A. Bazzan and F. Kluegl, editors, *Multiagent Architectures for Traffic and Transportation Engineering*. Springer, 2009. to appear.

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

[140] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, 2004.

[141] K. Tumer and D. Wolpert. The theory of collectives. *Collectives and the Design of Complex Systems*, 2004.

[142] H. Valizadegan, R. Jin, and S. Wang. Learning to trade off between exploration and exploitation in multiclass bandit prediction. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

[143] K. Verbeeck, M. Peeters, A. Nowe, and K. Tuyls. Reinforcement learning in stochastic single and multi-state games. *Adaptive Agents and Multiagent Systems II, Lecture Notes in Artificial Intelligence*, 2005.

[144] T. Walsh, S. Goschin, and M. Littman. Integrating sample-based planning and model-based reinforcement learning. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2010.

[145] C. Watkins. Learning from delayed rewards. In *PhD Thesis, University of Cambridge, England*, 1989.

[146] C. Watkins and P. Dayan. Technical note q-learning. *Machine Learning*, 1992.

[147] S. Whiteson, M. Taylor, and P. Stone. Empirical studies in action selection for reinforcement learning. *Adaptive Behavior*, 2007.

[148] E. Wiewiora. Efficient exploration for reinforcement learning. *Thesis, University of Pittsburgh*, 2004.

[149] S. Williamson, E. Gerding, and N. Jennings. Reward shaping for valuing communications during multi-agent coordination. *In Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[150] S. Witwicki and E. Durfee. Towards an unifying characterization for quantifying weak coupling in dec-pomdps. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[151] D. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. *In Advances in Neural Information Processing Systems*, 1999.

[152] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.

[153] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons Ltd, 2002.

[154] E. Yang and D. Gu. A survey on multiagent reinforcement learning towards multi-robot systems. *Proceedings of IEEE Symposium on Computational Intelligence and Games*, 2005.

[155] C. Zhang. Scaling multi-agent learning in complex environments. *Dissertation*, 2011.

[156] C. Zhang and V. Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[157] C. Zhang and V. Lesser. Coordinated multi-agent learning for decentralized pomdps. In *Proceedings of the Seventh Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM)*, 2012.

[158] K. Zhang and W. Pan. The two facets of the exploration-exploitation dilemma. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 2006.