A GAME THEORETIC FRAMEWORK FOR TEMPORAL AND AGENT ABSTRACTIONS IN MULTIAGENT LEARNING

by

DANIELLE M. CLEMENT

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
May 2016

Abstract

A GAME THEORETIC FRAMEWORK FOR TEMPORAL
AND AGENT ABSTRACTIONS IN
MULTIAGENT LEARNING

Danielle M. Clement, Ph.D.

The University of Texas at Arlington, 2016

Supervising Professor: Manfred Huber

A major challenge in the area of multiagent reinforcement learning is addressing the issue of scale as systems get more and more complex. Systems with multiple agents attempting to explore and learn about the world they inhabit concurrently require more time and resources to learn optimal responses. Increasing the number of agents within a system dramatically increases the cost to represent the problem and calculate the solution.

Single agent systems address part of the scaling problem through the use of temporal abstractions, or mechanisms to consider a sequence of many actions as a single temporal “step”. This concept has been applied in multiagent systems as well, but the work is predominantly focused on scenarios where all of the agents work together towards a common goal. The research presented in this dissertation focuses on situations where agents work to achieve independent goals, but need to collaborate with other agents in the world periodically in order to achieve those goals. To address the issue of scale, this dissertation presents an approach to represent sets of agents as a single entity whose interactions in the world are also temporally abstracted. Through implementation of this framework, agents are able to make game-theoretic decisions via stage games of significantly reduced dimensionality.

The first focus of this dissertation is on defining the mechanism that agents within an agent set (or coalition) use to interact amongst themselves. A key contribution of this research is the application

of temporally extended actions, called options, to a multiagent environment. These multiagent options serve as the core behaviors of sets of agents. Those behaviors are assessed in terms of standard multiagent learning techniques. Also presented are varied approaches to form and dissolve coalitions in a semi-cooperative environment.

After establishing the key properties required to participate in a coalition of agents, this dissertation focuses on the interactions between coalitions of agents. Another key contribution of this research is the structure developed to game-theoretically select coalitions in which to participate, as well as to represent the rest of the world as abstracted sets of agents. This structure is defined and its performance is evaluated in a grid-world setting with decisions made by a centralized agent. Solutions are assessed for correspondence of the selected decisions as compared to those that would be made if all agents were represented independently.

Finally, this structure is refined and evaluated to support decision-making by independent agents in a distributed fashion. Performance of these algorithms is also assessed.

To date, the vast majority of research in this area has been cooperative in nature, hierarchically structured and/or focused on maximizing group performance. Focus on interactions between sets of agents is rare. This research is unique in the current field of multiagent reinforcement learning in its ability for independently motivated agents to reason about other agents in the environment as abstracted sets of agents within semi-cooperative environments.

Acknowledgements

I would especially like to express my gratitude to my advisor and committee chair, professor Manfred Huber. Thank you for your time, your willingness to collaborate and your commitment to seeing me through this process. Without your encouragement I would not have made it this far in the process. I would also like to thank the members of my committee, professor Jean Gao, professor Dave Levine, and professor Farhad Kamangar for their continued support throughout the development of my research. Thank you all for your willingness to help and for sharing your expertise with me.

I'd also like to recognize my coworkers, whose unwavering support made it possible for me to take time away from my day-job. Deserving of special recognition in this regard are Jonathan, Janet and Courtney.

In pursuing this degree, the support of my family has been invaluable. I would like to thank: my mother, for a decade (a lifetime, really) of novenas; my father, for providing a first-hand example of the pride and joy that comes of completing a long-sought degree; and Meagan, Pawel, Xavier and Stepney R., for the (much needed) Sunday afternoon study breaks.

Finally, the completion of this degree would not have been possible without Mike's unwavering encouragement and support. Thank you for riding this roller coaster with me.

April 21, 2016

Table of Contents

List of Illustrations

List of Tables

CHAPTER 1

Introduction

Determining an optimal course of action in a system with a large number of self-interested parties is complex for multiple reasons. The first is the sheer scope of the problem – as the number of individuals considered increases, the amount of information to consider increases exponentially [1] regardless of the immediate relevance of the information. The second reason for complexity lies in the word optimal itself. Refining a decision making process to provide an "optimal" solution can be time consuming and resource intensive and the benefit of such a pristine solution in real-world situations is unclear.

It does not appear that humans share this limiting adversity to large scales – otherwise no one would ever attend parties or football games. How does the human thought process account for the vast and varied hopes, dreams, and resultant behaviors of other humans without drowning in information? Social psychology points to stereotyping (or social categorization) as one mechanism humans employ to forecast the behavior of others based on previous experience [2], [3].

Consider, for example, a large, crowded party. Upon entering the party, it is unlikely that you consider each person as an individual – rather, you mentally group people based on proximity (the people by the bar), previous relationships (people I don't know), or some other valuable-to-you criteria. This allows you to determine a course of action without becoming overwhelmed by information – the aggregate behavior of the individuals is predicted based on their membership in an assigned-by-you group. Once the grouping has been established, your attention can focus primarily on the individuals in your group and (potentially) the other groups with which you interact or are interested in interacting.

What would a structure like this look like from a game theoretic/multiagent reinforcement learning perspective? Can intelligent agents successfully abstract details of individual behavior and regard other agents as groups on an ad hoc basis?

Our research focuses on two phases: first, determining a mechanism for a set of agents to coordinate together in environments where they are independently motivated, and second, scaling

that mechanism to allow agents to use ad hoc teams of agents when suitable to achieve their individual goals, and also to use these teams to abstract the behavior of other agents.

1.1 Motivation and Objective

As the world in which agents interact grows more complex, the amount of information that must be maintained increases exponentially. Bellman [1] termed this the "curse of dimensionality", which means that as the number of state variables increases, the number of states increases exponentially (an example is shown in Figure 1.1). In complex environments, multiagent systems are significantly impacted by the curse of dimensionality, as maintaining information about the state of other agents in the system increases not only with the complexity of the environment, but also with the number of other interacting entities.

Current research has focused on managing the curse of dimensionality through abstraction, most generally the abstraction of state information or the abstraction of time. We assert that it is also possible to address the scaling issue associated with a large number of agents by abstracting those

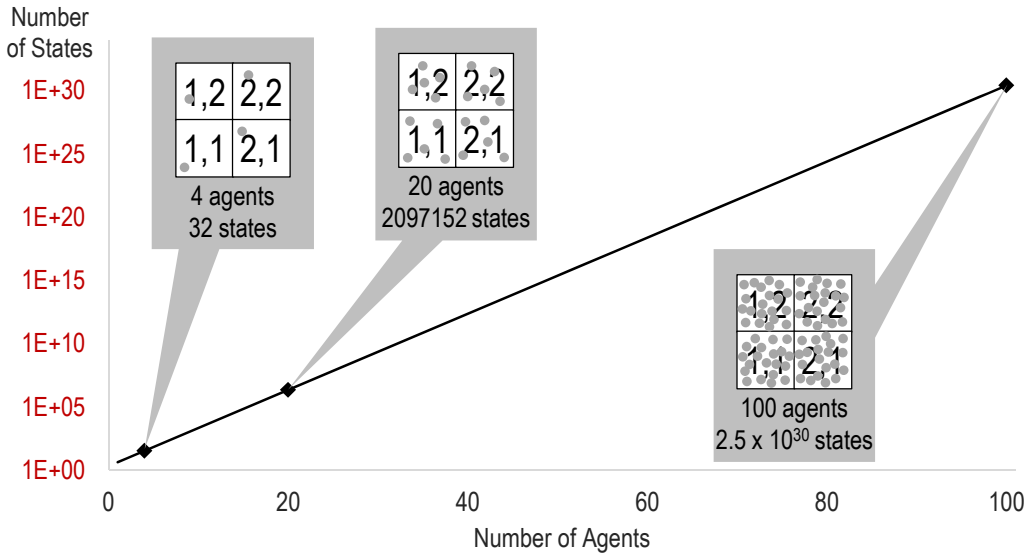

Figure 1.1 A Notional Depiction of the "Curse of Dimensionality".

Even in a small 2x2 grid world where the only relevant features are the position of the agents, as the number of agents (and thereby the number of state variables) increases, the number of states increases exponentially.

agents into groups, and considering only information about those groups which are relevant to the current situation.

To date, research in regard to social abstraction (or considering groups of individuals as a single aggregate entity) has focused mainly on abstracting groups into hierarchies, where individuals at higher levels of the hierarchy are responsible for informing the decisions made by individuals lower in the hierarchy, or are responsible for making those decisions in a centralized way. Predominantly, those structures are focused on maximization of a single social utility (a take-one-for-the-team worldview). Where agents do maintain independent utility functions, the focus also is on common utility or goals, or purely competitive scenarios. Where ad hoc teams are considered, they also tend to focus on a shared value or goal for all agents.

1.2 Overview

The research presented in this dissertation focuses on social abstraction of agents into ad hoc teams via coordinated extended time scale actions. A conceptual view of this approach is illustrated

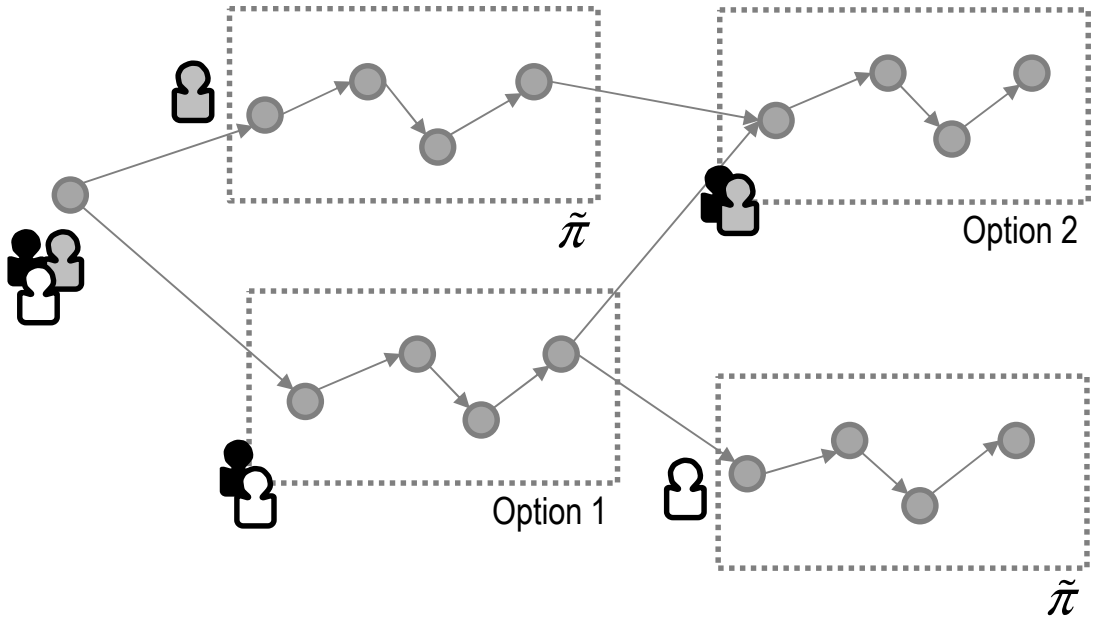

Figure 1.2 Overview of Proposed Approach

In this scenario, multiple agents interact with each other both within teams and between teams to achieve their desired goals.

in Figure 1.2. Agents in our system form coalitions, which execute collaborative sequences of actions (multiagent options) to achieve some well-defined subgoal. As opposed to fully cooperative scenarios, the collaborative sequences are represent common approaches to achieve mutually beneficial outcomes in non-cooperative coalitions. While executing these multiagent options, coalitions interact with one another, and at the conclusion of their interaction, they dissolve, only to form new coalitions across the set of agents. This approach forms abstractions in terms of the intra-coalition strategies through the use of multiagent options as well as in the inter-coalition strategy formation by reducing the decision problem to an inter-coalition equilibrium. Imposing a bound on the latter dimensionality represents the situation where parties can only reason explicitly about a limited number of entities (agents or coalitions) and reasoning about the details of agents within coalitions has to occur implicitly through the multiagent options. In this form, this approach aims at reducing complexity in non-cooperative games and improving scalability of such systems through social abstraction.

The reason for combining temporal abstractions and agent set abstractions is that when applying social abstractions to a system of agents, especially when attempting to address the curse of dimensionality, considering groups of agents for only a single time step is often not beneficial. Since groups of agents do not necessarily interact with other groups of agents on a single primitive time scale, it is important to also consider temporal abstraction methods, such as options [8]. Another potential benefit of considering groups of agents as a single entity would be to reduce state information to no longer consider all of the individual players. However, this research is primarily focused on the mechanisms of social abstraction, namely how coalitions form, dissolve and learn. While we do address temporal abstraction through the use of multiagent options as coalition primitives, we will leave state abstraction techniques to future work.

1.3 Summary of Key Contributions

Key contributions presented in this document include:

Definition of a game-theoretic approach to reducing system dimensionality through multiagent options. Multiagent options are the extended time-step actions which multiple agents elect to execute simultaneously. They provide a means to coordinate among self-interested agents to achieve subgoals, while not requiring any a priori development of a hierarchical structure. Selecting a multiagent option to execute enables an agent to make more robust predictions about the behavior of other agents within a group of agents.

Extension of that approach to define how sets of agents interact with each other.

In the two player domain there are really only two possible social choices – execute primitive actions as an independent agent (a coalition of one), or join forces with the only other agent to work collaboratively. Moving to worlds with higher numbers of agents allows for analysis and experimentation in the interactions between teams of agents.

Definition of mechanisms to reduce the amount of information that must be maintained by an individual agent in order to reason about sets of agents.

There is a learning benefit to reasoning about coalitions in a reduced game space, but there is also benefit in not having to maintain and update expectations for all the other entities in the system. This dissertation defines some mechanisms to allow for an agent to maintain the current expectations about the other agents at an aggregate (not individual) level.

Experimentation on distributed selection of equilibria in support of this structure.

While much of the work described in this dissertation selects equilibria as a centralized agent, we also discuss some challenges and propose a solution to manage the complexity that occurs when decision making is decentralized.

1.4 Roadmap to the Remainder of this Document

The remainder of this document is structured as follows:

Chapter 2 introduces the reader to the underlying technical concepts required for an understanding of the contributions of this work, as well as formalizing notation that will be used throughout the dissertation.

Figure 1.3 A Roadmap to the Remainder of This Document

The document progresses from discussion about a single coalition of agents and the behavior within to eventually describe a system where multiple coalitions can interact.

Chapter 3 reviews other related work in the field that has influenced this research and provides a relevant context for the contributions described in this dissertation.

Chapter 4 outlines the concept of multiagent options as a mechanism to coordinate behaviors of agents within a given team.

Chapter 5 addresses work done with multiagent options in a two player domain, including discussions of benefit to learning time and preliminary analysis of concepts that will require attention in domains with larger numbers of participants.

Chapter 6 extends the work presented on multiagent options (Chapter 4) to describe a framework for considering multiple agents as a single entity game theoretically.

Chapter 7 presents an experimental evaluation of the approach defined in Chapter 6.

Chapter 8 extends the previously described approach to systems of agents where the decision is made in a distributed fashion.

Chapter 9 describes key conclusions and summarizes proposed future work in this area.

CHAPTER 2

Technical Background and Notation

This work focuses on two main areas of artificial intelligence research, reinforcement learning and game theory. This chapter describes several foundational concepts helpful in understanding the remainder of this document, namely reinforcement learning and associated algorithms and key concepts in game theory.

2.1 Reinforcement Learning

Reinforcement learning [4], [5] introduces the concept of an agent, which is a learning entity with the ability to sense characteristics of an environment in which they are trying to achieve a goal. The standard visualization of a learning agent is shown is Figure 2.1. An agent interacts with an environment by receiving information about the environment (observations) and using those observations in a decision algorithm to determine an action to take in the environment. That action may influence the environment, resulting in new observations and driving new actions. While not all agents employ learning mechanisms to determine an appropriate action (for example, an agent that selects all actions randomly would not need to perform a learning step), the steps of updating

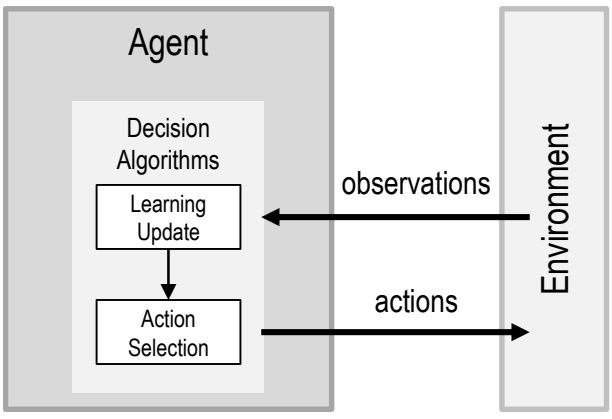

Figure 2.1 Model of an Agent in an Environment.

Agents receive information from their environments via sensors (or observations) and use that information to make decisions and select actions in pursuit of a goal.

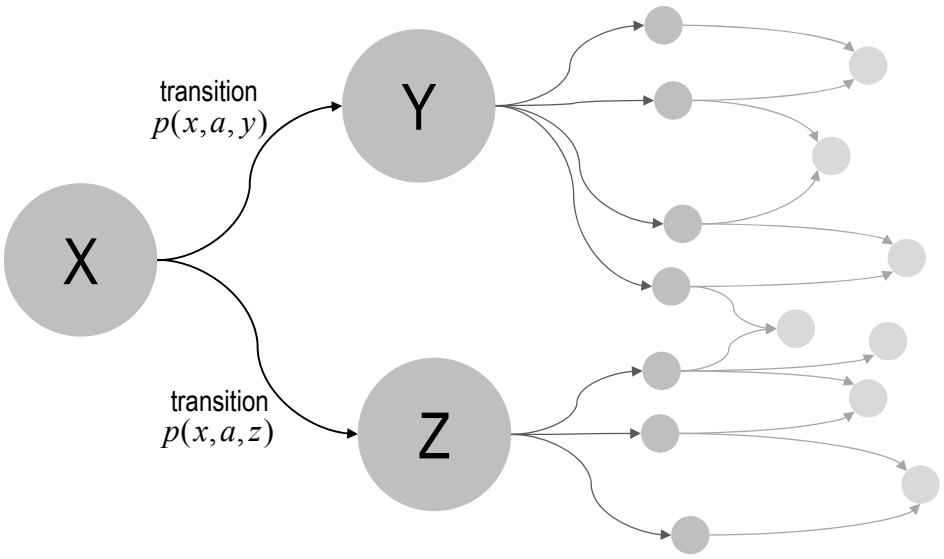

Figure 2.2 A Conceptual view of a simple Markov Decision Process

A simple Markov decision process is shown above. In this process, agents starting in state X and choosing action a transition to states Y and Z per defined transition probabilities.

expectations and plans based on observations and using that update to refine the selected action is prevalent in the following discussion.

In reinforcement learning algorithms, agents attempt to learn an optimal (meaning resulting in the highest value outcome) sequence of actions to execute based on feedback from the environment in the form of rewards.

2.1.1 Markov Decision Processes

A key concept in reinforcement learning is that of the Markov Decision Process, or MDP (conceptually represented in Figure 2.2). MDPs model the environment through a series of states, actions, transitions and rewards. More formally, an MDP is a tuple $\{S,A,T,R\}$ where S is a set of states, A is a set of actions, $T(s,a,s'),T:S\times A\times S\rightarrow[0,1],$ is the probability of transitioning to state s' from state s when action a is selected, and $R(s,a)\rightarrow\mathbb{R}$ is the reward received when an agent selects action a in state s. MDPs provide a structure to formally reason about a given problem. They are termed Markov because of the property that the effect of an action in any given state depends solely on the properties of that state, and not on any previously visited states.

Given the set of potential states, the model of transitions between states, and the rewards associated with selecting an action in a given state, it is possible to define an action choice associated with each state. A policy $\pi : S \times A \rightarrow [0,1]$ represents the probability that an agent will select action a in state s, and the designation π^* refers to the policy that will result in the highest possible expected outcome for the agent. Through the remainder of this section, for ease of notation, we will assume a deterministic policy $\pi(s)$ where in each state s, there exists a single action $a \in A_s$ such that $\pi(s,a) = 1$ and for all other actions $a' \in A_s$, $\pi(s,a') = 0$.

2.1.2 Utility

When determining the best actions for agents to take, it is helpful to understand the potential value from this point forward. The utility of a given state s can be thought of as the accumulation of reward values from that state onward. Since the overall expected reward values depend on the sequence of actions selected (or policy, as defined above), we define utility in terms of a given policy as in Equation (2.1), where $0 \leq \beta \leq 1$ represents a discount factor. The discount factor allows us to apply less value to the rewards of actions in the distant future than we do to actions in the near future. A value of β very close to 0 represents a very myopic view of the future, while a value of β very close to 1 represents a much longer relevant time horizon.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \beta^t R(s_t) \middle| \pi, s_0 = s\right] \tag{2.1}$$

This notion of state utility allows agents to evaluate actions based on not only the immediate return of single-step choices, but on the long term implications and value of those choices. To that end, agents can calculate the utility of each state based on the immediate reward of the current state, and the greatest possible utility of the states immediately following it, as in Equation (2.2).

$$U(s) = R(s) + \beta \max_a \sum T(s,a,s')U(s') \tag{2.2}$$

The policy for any given state s resultant from this update is defined as in Equation (2.3). This is the policy that will select the action that maximizes the expected utility in future states.

$$\pi(s) = \arg\max_a \sum_{s'} T(s,a,s')U(s') \tag{2.3}$$

2.1.3 Value and Policy Iteration

Agents with no information about the utility of states, but with knowledge of the transition properties of the environment, can iteratively apply this utility assessment to determine the utility of states via a method called value iteration, shown in Equation (2.4).

$$U_{t+1}(s) = R(s) + \beta \max_a \sum T(s,a,s')U_t(s') \tag{2.4}$$

This calculation requires the agent to maintain a table of current utility values for each state. Those values are defaulted to a nominal value. At each time step and for each state, the agent can calculate the best action based on current utilities and combine that with the reward in the current state to update the utility value table. Note that this calculation requires the agent to know the transition probabilities between states or the *model* of the environment. Learning algorithms that do not require this information are called *model-free*.

Policy Iteration is another model-based learning method that splits value iteration into two steps -- alternating between the assessment of the utility of the current policy and the update of that policy to be the optimal based on the current set of utilities. The policy is updated by selecting the action that results in the highest expected utility for future states (per Equation (2.3)), and the utility of the current policy is assessed based on Equation (2.5). The primary difference between policy and value iteration is that policy iteration is focused on converging to an optimal policy regardless of the convergence of the supporting utility values, whereas value iteration is focused on converging utility values to a maximum, and thereby resulting in an optimal policy. Because utility convergence is

really a means to an end to ensure policy convergence, policy iteration can converge more rapidly than value iteration.

$$U_{t+1}(s) = R(s) + \beta \sum T(s, \pi_t(s), s') U_t(s') \tag{2.5}$$

2.1.4 Q-Learning

Q-learning, established by Watkins [6] in 1989, is a process of learning the values of actions in every state (called Q-values) and using that information to determine the best action in a given state. These Q-values are updated based on experience and do not require any knowledge of the transition probabilities between states. The update function is shown in Equation (2.6).

$$Q_{t+1}(s_t, a_t) = (1-\alpha_t) * Q_t(s_t, a_t) + \alpha_t \left[R_t + \beta \max_a Q_t(s_{t+1}, a) \right] \tag{2.6}$$

where α_t is a learning rate which balances the influence of current information against new updates, β represents the discount factor as described previously, and R_t is the learner's reward at time t. The Q-values stand-in for weighted expected utility values in the learning mechanism, allowing the agent to learn the best reward based on actual outcomes, however, sufficient experience of the state space and action choices is required to converge to an optimal solution.

2.1.5 SARSA

While Q-learning updates the current Q-values based on the best possible next-step action, SARSA [7] is a learning mechanism that updates Q-values based on the actually selected next-step action a_{t+1}. This requires the agent to be aware of the current state, the current action, the reward received, as well as the next state and the next action. The update function is then as described in Equation (2.7).

$$Q_{t+1}(s_t, a_t) = (1-\alpha_t) * Q_t(s_t, a_t) + \alpha_t\left[R_t + \beta Q_t(s_{t+1}, a_{t+1})\right] \quad (2.7)$$

2.1.6 Options

The mechanisms discussed to this point have been concerned with selecting the best single-step action to take in order to maximize long-term utility. Once policies over these single-step actions have been learned, it is possible to abstract these low-level behaviors into higher order, more complex sequences of actions called *options*.

Options, as described by Sutton, Precup and Singh [8], provide a mechanism in single agent systems to extend actions over multi-step time scales. Formally, options $\{\pi, \tau, I\}$ are comprised of:

- a policy $\pi : S \times A_p \rightarrow [0,1]$ where S is the set of all states and A_p is the set of primitive actions, and where π represents the probability the primitive action $a \in A_p$ is selected in a given state $s \in S$;
- a termination condition $\tau : S \times A_p \rightarrow [0,1]$ where S is the set of all states and A_p is the set of primitive actions, where τ represents the probability of terminating the option;
- an initiation set $I \subseteq S$, which represents the set of states in which an option is valid to start.

When in an initiation state, an agent has the ability to select an option associated with that initiation state. Once an option has been selected, an agent will follow the policy associated with that option until that option terminates per τ. Primitive, or single-step, actions can also be defined as options, and combined with the SMDP work of Bradtke and Duff [9], the Sutton, Precup and Singh research defines policies over options, thereby enabling temporal hierarchies of behavior. Using this hierarchical SMDP RL approach has been shown to have the potential to significantly accelerate learning and to allow for larger problems to be learned than could be achieved using only primitive actions. [10]

2.1.7 Multiagent Reinforcement Learning (MARL)

Multiagent reinforcement learning is complex for many reasons, including the "curse of dimensionality" (the state space grows exponentially when characteristics of other agents must be taken into account) and the fact that as individual agents perceive and influence their environment, other agents in the system are doing the same, which can perturb and delay the learning process (learning times are delayed due to inconsistent execution). Despite these complexities, multiagent RL shares much with single agent RL, and many of the RL algorithms previously discussed can be applied to MARL. Further discussion of MARL techniques is summarized in Chapter 3.

2.2 Game Theoretic Approaches

Game theory focuses on the analysis of competition and/or cooperation between different entities. While reinforcement learning has its roots in single agent learning, game theory is inherently multiagent. Game theory looks at the interactions between players as games they play, where each player makes a decision between potential actions based on the potential payoff received across players.

The following section leans heavily on the notation described in Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations [11]. A classification system of games is depicted to the right in Figure 2.3, with each of the subtypes described in the following sections. A normal form game describes an interaction between multiple parties, where each party can simultaneously chose one of several actions, and a reward (or payoff) is given to each agent based on the set of actions selected by all agents. A repeated game is a normal form game played for multiple iterations. A stochastic game is a game where the actions selected by the participating agents result in another game to be played (in addition to the incremental payoffs). Markov Decision Processes (previously discussed) can be considered a special case of stochastic games where only one player selects actions. Repeated games can be thought of as a special case of stochastic games where all action choices result in transitioning back to the same state. Each of these types of games will be discussed in detail in the following sections.

2.2.1 Normal Form Games

A normal form game is defined as $\{N, A, u\}$ where N is a finite set of n players, $A = A_1 \times \ldots \times A_n$ is the set of action vectors per player, and the utility $u = (u_1, \ldots, u_n)$ where $u : A \to \mathbb{R}$ is the vector of payoffs, or rewards, for each player. Two player normal form games are typically depicted in a matrix as shown in Figure 2.4, where each player's action choices are displayed across the axes, and the payoffs for each player are shown in (row, column) format.

A strategy π defines the probability with which a given player should choose an action. When a given strategy contains one and only one action with probability greater than 0, that strategy is

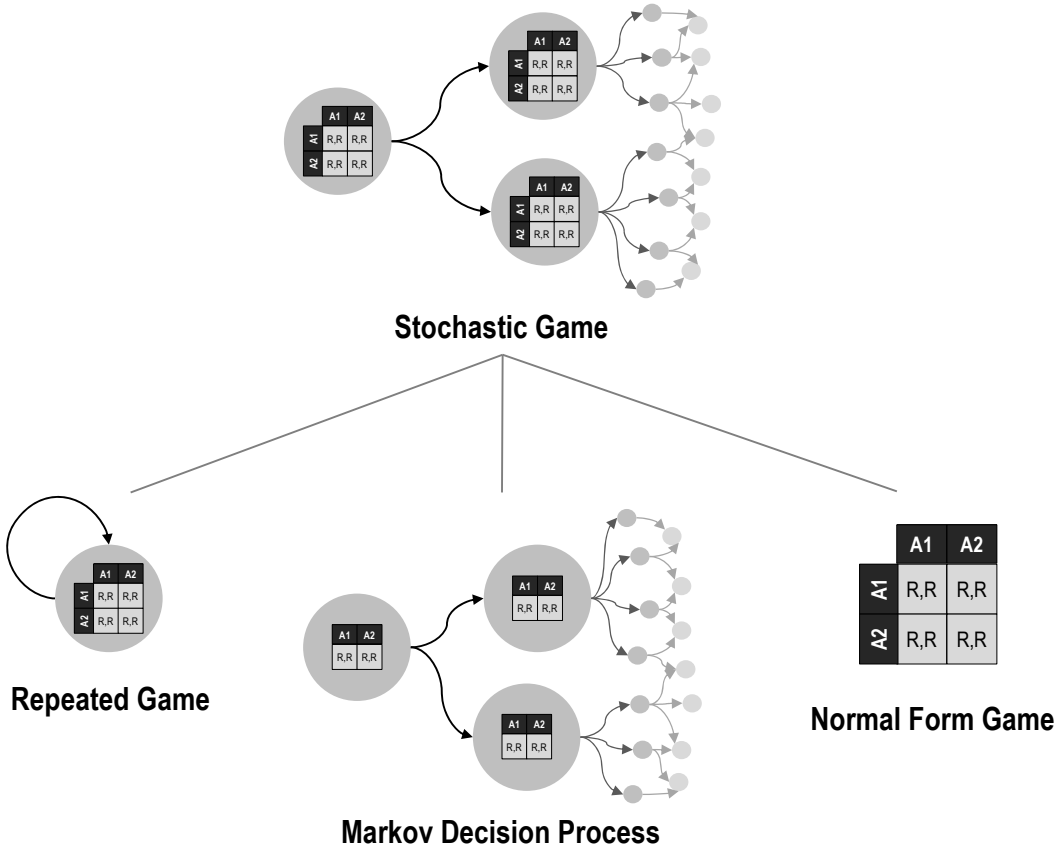

Figure 2.3 Overall Taxonomy of Games

Stochastic games are the game theoretic equivalent of Markov Decision Processes, in fact, MDPs can be considered a stochastic game with only one player. Special forms of stochastic games also include the repeated game, which is a stochastic game where all actions transition to the same game, and the normal form game, which is a stochastic game where all actions transition to the end state (one shot).

	A	B	C	D
X	(x, a)	(x, b)	(x, c)	(x, d)
Y	(y, a)	(y, b)	(y, c)	(y, d)
Z	(z, a)	(z, b)	(z, c)	(z, d)

Figure 2.4 A Normal Form Game in Matrix Format.

Player actions are shown in the row and column headers, and the payoffs for those action combinations are shown in the cells, in (row, column) order.

termed a *pure* strategy. Strategies that probabilistically combine action selections are called *mixed* strategies. The support of a strategy is the set of all actions where the probability of executing that action is greater than 0 in that strategy.

The utility of a strategy for a given player P_i follows Equation (2.8). This equation is given for the two player case, but can be generalized to support more than two players.

$$U_{P_i}(\pi) = \sum_{a_j \in A_i} \rho(a_j \mid \pi) * \sum_{b_k \in A_{k \neq i}} \rho(b_k \mid \pi) * u_i(a_j, b_k) \tag{2.8}$$

2.2.2 Best Response and Nash Equilibria

A best response to other players' strategy is the strategy which, if an agent were to play anything else, their ultimate individual utility would decrease. This would be a wonderful property if the strategies of the other players were known in advance, however, this is rarely the case. A Nash Equilibrium[12] is a set of strategies (or strategy profiles) where **each** player's utility would decrease if they (and only they) deviated from this strategy. In a Nash Equilibrium, every individual agent's strategy is a best response to the other agents' strategies. Nash Equilibria can be mixed or pure strategies, and every game has at least one. While Nash Equilibria are inherently stable, they are not necessarily the solution resulting in the highest payoff, as is shown in the oft-cited prisoner's

	Stay Silent	Betray
Stay Silent	(-1, -1)	(-5, 0)
Betray	(0, -5)	(-10, -10)

Figure 2.5 The Matrix Game For Prisoner's Dilemma

The matrix game for prisoners dilemma has a Nash Equilibrium at {Betray, Betray}, even though there is a better solution for both players at {Stay Silent, Stay Silent}.

dilemma example (Figure 2.5). Also, if there happen to be multiple Nash Equilibria for a given game, those equilibria may not have the same utility among themselves or for all players. For this reason, when each player selects an equilibrium strategy to play from multiple options, consideration must be given to how the players will select a common equilibrium.

2.2.3 Stochastic Games

Stochastic games combine the principles of game theory and Markov decision processes (MDPs). Specifically, stochastic games are defined by $\{S,N,A,P,R\}$ where S is a finite set of games, N is a finite set of n players, $A = A_1 \times ... \times A_n$ is the set of action vectors for each player, P is the transition function $P(s,a,s') \rightarrow [0,1]$, or the probability of transitioning from state s to state s' given the actions a , and $R = r_1,...,r_n$ is the reward where r_i is the single step reward for player i . To compute the overall payoff for a stochastic game, we compute the discounted reward for a given player i as shown in Equation (2.9).

$$U^{\pi}(i) = \sum_{j=1}^{\infty} \beta^j r_i^{(j)} \tag{2.9}$$

In the previous Equation, $0 \leq \beta \leq 1$ represents the discount factor, or the amount that a player cares about payoffs in the distant future. The value $r_i^{(j)}$ represents the payoff to player i in stage game j . This utility value over the discounted horizon becomes the payoff for player i in the stochastic game.

2.2.4 General Sum Stochastic Games

Stochastic games are considered general-sum if the sum of the payoffs to each player do not equal zero. Zero-sum stochastic games are considered purely competitive games (one player's loss is another player's gain), whereas general sum stochastic games cover the remainder of all stochastic games, from the strictly cooperative to games that have both cooperative and competitive aspects.

2.2.5 NashQ

NashQ is a reinforcement learning algorithm that extends Q-learning to the game-theoretic multiagent domain. Since in these domains greedy utility maximization by each agent is no longer a viable solution, NashQ learns instead a game-theoretic equilibrium strategy. In particular, NashQ learning for multiagent systems [13], uses the Nash Equilibrium as the update feature for Q-learning as opposed to the max. This was chosen as a mechanism to address the fact that learning in multiagent systems is challenging due to the uncertainty associated with other agents' explorations and that an agent who possesses some knowledge about the choices, benefits or rewards of other

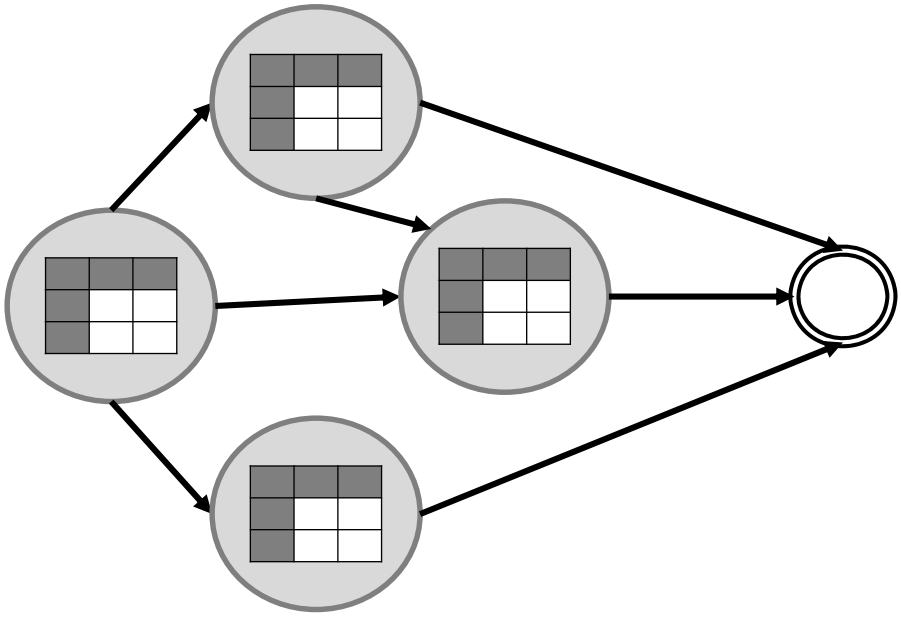

Figure 2.6 Conceptual Depiction of a Stochastic Game

Stochastic Games include transitions to new states much like MDPs, however, it is assumed that at any given state, the players in the system play a game based on the rewards in that state.

agents can improve their learning by making decisions informed by that knowledge in addition to reasoning about their own behavior.

A NashQ learner learns in a distributed fashion and reasons that the other agents within a system will behave rationally (in a game-theoretic sense) in future interactions, and therefore selecting an equilibrium behavior will prove more advantageous in semi-cooperative games than selecting an independent max behavior.

NashQ, formally, extends the traditional Q-value update equations for reinforcement learning to stochastic multiagent games with respect to a given player i as:

$$Q^i_{(t+1)}(s_t, a^1, ..., a^n) = (1-\alpha) * Q^i_t(s_t, a^1, ..., a^n) + \alpha_t[R^i_t + \beta NashQ^i_t(s')] \qquad (2.10)$$

where $NashQ^i_t(s')$ is the expected payoff of the Nash equilibrium strategy of the stage game associated with the next state Q-values, t and $t+1$ represent timesteps. Selection of that equilibrium strategy consistently across agents is one of the challenges of NashQ. For the initial implementation, Hu and Wellman [13] selected the first equilibrium.

2.3 Summary

This chapter described the core definitions, algorithms and techniques of multiagent systems and multiagent learning. The following chapter will expand upon this foundation to discuss relevant research that has extended these concepts and will provide context for this work as a part of the current body of knowledge in multiagent reinforcement learning.

CHAPTER 3

Related Work

3.1 Introduction

Bellman [1] defined the *curse of dimensionality* as the property that computational requirements for learning grow exponentially with the number of state variables considered. This problem is particularly thorny in multiagent domains, as the need to represent information about other players drives the need to include that information as additional state variables. For this reason, much research has been done to date in abstracting state and time to address the curse of dimensionality in single and multiagent state spaces.

3.2 Multiagent Systems and Large Domain State Spaces

For the purposes of this discussion, the concept of dealing with an overly detailed world through abstraction is divided into three major areas, with current contributions summarized in each. Those three areas are: *time abstraction*, or the representation of behavior on multiple time scales; *state abstraction*, or the collapsing of state information into a subset of states relevant to the current problem, and *social abstraction*, or the representation of multiple agents in the system as a single entity or team. For the purposes of this categorization, systems where multiple agents learn simultaneously are classified as multiagent learners. Systems where only one agent learns at a time, such as Stone, Sutton and Kuhlman [14] are classified as single agent learners, even when there are multiple agents in the environment. Panait and Luke [15] divide the world of cooperative multiagent learners into two main types: *team learners*, who use a single agent to learn a policy for a group, or team, of agents, and *concurrent learners*, who learn policies independently. For the purposes of this work, we will focus mainly on concurrent learners, but will highlight a few notable team learning examples. Note also that Panait and Luke's classification scheme focused solely on cooperative agents, whereas this research focuses on semi-cooperative agents. The research overall is summarized in Figure 3.1.

Specifically focused on this area of interest, social abstraction is wholly related. The concept of representing a group of agents as a single coherent entity is foundational to this research. Abstracting agents into groups is an emerging area of research, especially in semi-cooperative domains. Work in time abstraction is similarly related, as it follows that reasoning about groups of agents requires reasoning about higher-level, joint actions (representing an offensive play in soccer, not an individual player's pass, for example). Without this level of abstraction, the number of relevant players may collapse, but the action set would increase to account for their individual actions. State abstraction is related in a different way. In order for agents to accept roles within a

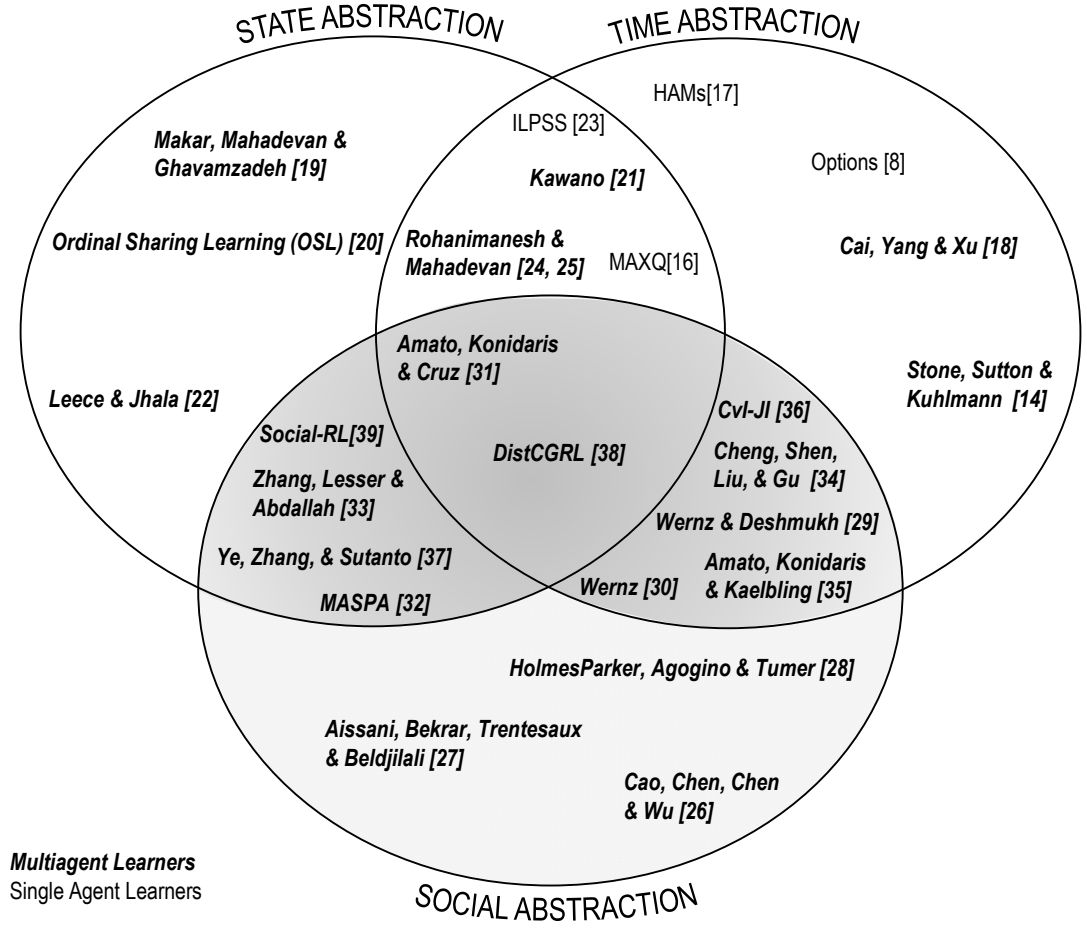

Figure 3.1 Summary of Related Work

Dealing with the "curse of dimensionality" through abstraction is actively being pursued on multiple fronts, including research into representing multiple timescales hierarchically or simultaneously, abstracting the behavior of agents into team structures, and reducing state spaces through abstraction.

coalition and to perform those roles, some level of state abstraction needs to be in place, so that policies over the states can focus on relevant state details (the position of the agent in role 1, as opposed to Player 1's position). This is analogous to some of the work done in transfer learning of policies.

3.2.1 Time Abstraction

Time abstraction is critical to this research, as being able to reason about the environment at multiple time-scales is required to draw conclusions about the long-term benefits of coordination behaviors.

Options [8] are discussed extensively in the previous chapter. They abstract actions temporally into options, which can extend over multiple timesteps. Using an SMDP framework, options can be treated as actions, meaning that an agent can create hierarchical policies over options and begin to develop sub-routines tied to subgoals. Other frequently extended temporal abstraction methods include MAXQ [16], which decomposes the world into a set of subtasks and then computes a policy over the lower-level subtasks, and Hierarchical Abstract Machines (HAMs) [17], which specify partial policies through finite state automata which then call lower level machines.

Research that has been done to extend temporal abstraction into the multiagent domain has primarily been focused on pure cooperative tasks. Cai, Yang & Xu [18] integrated options with MAXQ to support learning in a multiagent robotics domain. Their work, however, was purely cooperative, and agents cooperated through MAXQ subtasks. In addition, Stone, Sutton & Kuhlmann [14] used options to select actions in a multiagent setting for a soccer keepaway task. While this is a multiagent problem, only the agent with the ball learns any behavior and the options used focus on that agent's behavior. Like Cai, Yang and Xu, all players share the same goal (offense only), so the task is purely cooperative.

3.2.2 State Abstraction

We consider state abstraction to be the process of identifying features of the state that can be disregarded or aggregated in some way, thereby (a) reducing the state space and (b) ensuring that

states that share common features can be used in similar ways for agents. In this way, MAXQ and its derivative algorithms also utilize some level of state abstraction. State Abstraction provides a valuable mechanism to partition the world, and is critical to our research here, as little to no benefit is anticipated to social abstraction (or considering a group of agents as a single entity) without the ability to also maintain limited information about characteristics of the individuals within the team itself.

Again, much of the work done in the multiagent domain focuses on purely cooperative tasks. Makar, Mahadevan & Ghavamzadeh [19] use a hierarchy of tasks to allocate work among agents. Their multiagent algorithm is used to enhance MAXQ to enable a group of agents to efficiently search a set of rooms. It is an inherently cooperative objective, and the state abstraction is due to the hierarchy of tasks requiring both local and global state information. Ordinal Sharing Learning (OSL) [20] focused on state abstraction in a grid computing environment, looking at decentralized job schedulers and how to minimize job time to wait. In their framework, agents know their local state and communicate some information about neighbors' values. Since the schedulers are working with shared resources, collaboration is inherent, but not explicitly programmed into the system. Agents maintain an independent utility, but the system is inherently cooperative. Kawano [21] automatically decomposes a MDP state space into a hierarchical space using SMDPs. Their system is also cooperative, and executes in a domain with multiple robots pushing a box around a maze. Agents collaborate on subtasks, but they never explicitly form teams.

On the other side of the spectrum, Leece & Jhala [22] seek to improve performance against unknown agents in purely competitive domains. Their system is turn-based and focuses on real time strategy games like Wargus, but in their system, all decisions are made globally.

Learning transfer is in some sense analogous to the type of abstraction we would like to obtain with roles – we would like to be able to represent the state of the world based on the state of the agents participating in a given multiagent option. Hawasly & Ramamoorthy's Incremental Learning of Policy Space Structure (ILPSS) [23] is a single agent algorithm focused on learning transfer and policy "snippet" abstraction to deal with new environments. They test their algorithm in a simulated

robot soccer game (the scenario is multiagent, but much like Stone's keepaway, only the agent with the ball learns at any given moment). Their algorithm leverages options to create the policy snippets. They also use the state space structure to determine which policy snippets to transfer.

Rohanimanesh & Mahadevan's [24][25] work with SMDPs focuses on determining termination schemes for concurrent option execution. Their algorithms can be applied to single agents executing tasks in parallel (walking and chewing gum) or to multiple agents working concurrently. This work is not focused on learning per se, but on characterizing the behavior of agents under various termination schemes. They test these algorithms in a grid world, and factor the state space based on the values needed to execute any given option.

3.2.3 Social Abstraction

Social abstraction is the primary focus of this research. Social abstraction is defined as the capability to consider a group of agents as a single decision making entity (or team). Much of the work today has been focused on either (a) hierarchical, pre-designed organizations that share a common goal or (b) explicitly competitive team structures, as in soccer. Our work focuses on ad hoc interactions between teams to achieve a short term individual goal for agents that are semi-cooperative.

In the extreme, Cao, Chen, Chen & Wu [26] abstracted other agents in that they treated all other agents as part of the environment and learned under that assumption. They used Friend-Q for a formation control task. Their mechanization worked because they assumed that the group reward is consistent with the individual reward, and therefore maximizing individual rewards would result in maximal group rewards. This assumption holds true for fully cooperative systems, but does not hold true for semi-cooperative environments.

Frequently, social abstraction takes the form of fixed hierarchical structures. Aissani, Bekrar, Trentesaux & Beldjilali [27] developed a job scheduling application for multiple agents satisfying the flexible job shop problem. Their mechanism, based on SARSA, divided agents hierarchically with communication between different levels of the hierarchy. The established hierarchy was fixed,

however, so there was no dynamic team formation in their approach, however, they did use individual agent rewards. HolmesParker, Agogino & Tumer [28] combined hierarchical organization and reward shaping methods to scale up a system of agents. They used Ɛ-greedy Q-learning to solve the Defect Combination Problem and compared experimental results for limited team structure and hierarchically organized teams combined with reward shaping algorithms, determining that the combination of the two methods outperformed all other methods applied in isolation. Though hierarchically organized, teams were controlled by a single agent and acted as extensions of that agent's sensor capability (single agent learner), and sought to optimize a shared utility.

Wernz & Deshmukh [29] coupled a hierarchical organization structure with extended time step actions and rewards. They included agent interdependence in the criteria that influence transitions and then calculated the independent utility per agent. They structured their system into an operational hierarchy for escalator repair. Team members had independent goals to optimize, and did so by sharing reward across levels of the hierarchy (typically down the hierarchy). The hierarchies were fixed, but the utility was not shared between agents and the collective. As a follow-on, Wernz [30] generalized the previous work to larger time scales and organizations. The hierarchies were still established and fixed in the follow-on work.

Amato, Konidaris & Cruz [31] use decentralized POMDPs and options to create decentralized cooperative behaviors in robotic systems (two agents work together to move a large box). Their work extends Dec-POMDPs and options focusing on a warehouse domain. Their work is purely cooperative in that all agents have a shared utility/goal. Teams of agents are organized hierarchically and single agent macro actions are selected in a coordinated fashion.

MASPA, or Multiagent Supervisory Policy Adaptation, by Zhang, Abdallah & Lesser [32] overlays a supervisory hierarchy onto agents to create local neighborhoods. State abstraction moves up the hierarchy to supervisor agents. MASPA uses weighted policy learner as an example algorithm in their distributed task allocation domain. Their system is purely cooperative with a single system

utility subject to maximization. MASPA uses predefined hierarchies, and assumes that decompositions within a hierarchy act independently, which is not the case for this research.

Rather than employing a fixed hierarchical structure, Zhang, Lesser and Abdallah [33] build a dynamic hierarchy of agents. They use supervisory steps to guide and control interactions and create “rules” and “suggestions” to coordinate and manage subordinate behavior. Their system is a multiagent system that extends Dec-MDPs for a distributed task allocation problem, however, their work is purely cooperative. Collaborations drive the algorithms that cluster the agents into teams, which are dynamically created based on “proximity”. These teams are explicitly hierarchical, but the higher-level supervisor is explicitly and algorithmically selected to minimize communication cost between agents.

When hierarchies are not being employed, frequently decisions are made by a single agent to the benefit of all participants (typically purely cooperative). For example, Cheng, Shen, Liu, & Gu [34] integrated options into MAXQ and focused on a purely cooperative trash collection task. Their system would only coordinate at coordination levels (similar to the work of Ghavamzadeh [19]). Amato, Konidaris & Kaelbling [35] extend decentralized POMDPs to include options as a coordination choice for each agent. Their work focuses on meeting in a grid, or navigating among moveable obstacles. As a Dec-POMDP, this a purely cooperative task with shared utility among all agents.

CvI-JI [36] is a hybrid approach between Belief-Desire-Intention methods and collective versus individual decision making. They focus on an extension of Dietterich’s [16] taxi world where taxi agents must collaborate together to make sure large parties arrive in time. It is a fully cooperative scenario. Organizationally, there is a “collective stratum” that represents all agents as a single agent. Each agent chooses to make a decision independently or to have the collective make the decision on their behalf. A characterization parameter for each agent determines their level of self-interest. Teams are dynamically structured. Note: there are no penalties for leaving the team, in fact, premature termination led to increased performance during learning.

Ye, Zhang, & Sutanto [37] look at Q-learning as a mechanism for who to collaborate with from a self-organizing teams perspective. They focus on cooperative network applications and their agents concurrently learn a trust model. Agents independently update the strength of relationships to handle task allocation problems. Agents are primarily self-interested (independent utility), but they collaborate together to their own benefit. Agents join and leave the system, so there is some abstraction of state, and agents self-organize into communities (either peer-to-peer or hierarchical) to better manage their workloads.

Some work has been done in the areas of ad hoc, collaborative teams. DistCGRL (distributed coordination guided RL) [38] is an algorithm proposed by Lau, Lee and Hsu which defines coordination constraints specifically to reduce the relevant state space. It leverages Dec-MDPs to support a tactical RTS and simplified soccer scenario. It is a cooperative algorithm where agents can access the local state space of their neighbors. The structure of accessing only neighbors' state information means that the agents form teams in an ad hoc, proximity based way.

Social-RL, developed by Sun, Ray, Kralik and She [39] is a mechanism that describes ad hoc social hierarchies and roles and how those constructs can inform reinforcement learning. They use decentralized Q-learning and Dec-MDPs for a multi-robot foraging task in Webots. Their environment is semi-cooperative, but they do not use game theory to model the interactions within teams, rather, their teams are formed through an assessment of team member's capabilities (physical endurance, etc) against the requirements of group tasks. They also do not address any interactions between teams.

3.3 Summary

This chapter provided an overview of current research into state, time and social abstraction in reinforcement learning. Key insights and trends from each of the works surveyed are represented in Table 3.1. The next chapter discusses our contributions to this domain in the area of multiagent options, coalitions and roles.

Table 3.1 Summary of Related Work

This table summarizes the current body of multiagent systems research, focusing on teams where agents are organized hierarchically.

Citation	*Single or Multiagent*	*Hierarchical or Ad Hoc?*	*Purpose*	*Cooperation Type*	*How Agents Join*	*How Agents Leave*
Amato, Konidaris, Cruz, Maynor, How, and Kaelbling, [40]	Multiagent	Hierarchical	Maximize common utility	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed
Amato, Konidaris, and Kaelbling [35]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	"Coordinated Controller"	"Coordinated Controller"
Cai, Yang, and Xu [18]	Multiagent	Hierarchical	Maximize common utility	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed
Cao, Chen, Chen, and Wu[26]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	Existence	The organization is fixed
Cheng, Shen, Liu, and Gu [34]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	"Coordinated Controller"	"Coordinated Controller"
Hawasly and Ramamoorthy [23]	Single Agent	Ad Hoc	Maximize common utility	Fully cooperative	State	State
HolmesParker, Agogino, and Tumer, [28]	Multiagent	Hierarchical	Maximize common utility	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed
Trigo and Coelho [36]	Multiagent	Ad Hoc	Maximize individual utility	Fully cooperative	"Coordinated Controller"	"Coordinated Controller"
Stone, Sutton, and Kuhlmann [14]	Single Agent	Ad Hoc	Maximize common utility	Fully cooperative	State (the player with the ball)	State (player no longer has ball)
Sun, Ray, Kralik, and Shi [39]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	Role/Capability Matching	Role/Capability Matching
Lau, Lee, and Hsu [38]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	Proximity	Proximity
Wernz [30]	Multiagent	Hierarchical	Maximize individual utility (pass rewards down hierarchy to motivate cooperation)	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed

Table 3.1 (cont) Summary of Related Work

This table summarizes the current body of multiagent systems research, focusing on teams where agents are organized hierarchically.

Citation	*Single or Multiagent*	*Hierarchical or Ad Hoc?*	*Purpose*	*Cooperation Type*	*How Agents Join*	*How Agents Leave*
Wernz and Deshmukh[29]	Multiagent	Hierarchical	Maximize individual utility (pass rewards down hierarchy to motivate cooperation)	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed
Ye, Zhang, and Sutanto [37]	Multiagent	Ad Hoc	Maximize common utility	Self-interested, but willing to collaborate to their own benefit	Proximity	Proximity
Zhang, Abdallah, and Lesser [32]	Multiagent	Hierarchical	Maximize common utility	Fully cooperative	Assigned to a hierarchy	The hierarchy is fixed
Zhang, Lesser, and Abdallah [33]	Multiagent	Hierarchical	Maximize common utility	Fully cooperative	Agents nominate a supervisor	Supervisors can trade individual agents
Sun, Ray, Kralik, and Shi [39]	Multiagent	Ad Hoc	Maximize common utility	Fully cooperative	Role/Capability Matching	Role/Capability Matching

CHAPTER 4

Multiagent Options: Defining Behaviors for Sets of Agents

This chapter focuses on a game-theoretic approach to reducing system dimensionality through multiagent options. Multiagent options are extended time-step actions (much like single agent options) which multiple agents elect to execute simultaneously. These options provide a mechanism to coordinate among self-interested agents to achieve subgoals, while not requiring any a priori development of a hierarchical structure. Selecting a multiagent option to execute enables an agent to make more robust predictions about the behavior of other agents within their "group of interest" (or coalition). Even for agents outside the coalition, multiagent options provide a means to abstract the behavior of multiple other parties into a smaller number of other coalitions and to determine a best response in that scenario.

4.1 Multiagent Options

To further explore the concept of multiagent options, consider the limited grid world shown in Figure 4.1. In this world, all agents are homogeneous and have three action choices $A = \{left, right, down\}$. Each move has a penalty of $R = -1$. Square D is the goal state and can only be reached if two (and only two) agents attempt to move down from square B in the same time step. Reaching square D will result in a reward $R = 10$. Attempts to move down to square D by more or fewer than 2

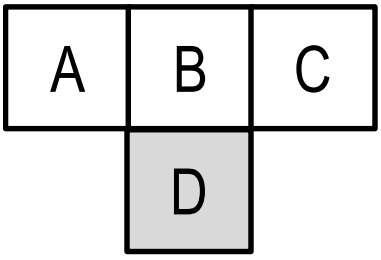

Figure 4.1 An Example Grid World Scenario.

In this scenario, agents have three actions, left, right and down. They can only enter square D from square B and only if two (and only two) agents both move down together. If they succeed in reaching square D, they both get 10 points. If they attempt to reach square D and fail, they lose 10 points. All actions cost agents 1 point.

agents will result in a reward of $R = -10$ for the agents that attempt to move. This small world contains 9 states when there are two agents present (square D is considered the terminal state and is not included). Each state has 9 possible transitions (based on the combination of both players' actions), resulting in an overall Q-table size of 81 values.

A multiagent option in this world would provide a coordinated mechanism for two players to collaborate and move down to square D together. The players would, independently and game-theoretically, be able to select if they wanted to work together as a team or select actions independently. These actions may be somewhat counterintuitive from an individual utility point of view. For example, consider the case when Player 1 is in square A and Player 2 is in square B. If both agents select to work together, they need to both be in square B at the same time, but there is no action for player 2 to remain in place. In this instance, a possible strategy would be for Player 1 to move {left, right, down} to end up in square D, while Player 2 moves {right, left, down} to also end up in square D (where the game ends). While this example world is a fully collaborative one, the same techniques can be used in semi-cooperative environments, as discussed further in Chapter 5.

Formally, multiagent options can be defined as a tuple $\{P, \pi_{1...n}, \tau, I\}$ where:

- $P \subseteq N$ is the set of participating players,
- $\pi_{1...n} : S \times A_{1...n} \rightarrow [0,1]$ is a set of policies (one for each participating player) where S is the set of all states, A_i is the of actions available to player i, and n is the number of participating players,
- $\tau : S^+ \rightarrow [0,1]$ is the probability that the multiagent option will terminate in state $s \in S^+$ and where S^+ is the set of all states including any termination states, and
- $I \subseteq S$ is the set of initiation states

The primary extension from the standard option framework (described in Section 2.1.6) is the shift from a single agent policy to a set of policies (one for each participating player) which jointly form a Nash equilibrium strategy for a given task represented by the multiagent option. When in an

initiation state, each player has the ability to select a multiagent option. If all participating players select the multiagent option, for as long as all participating players jointly pursue that option, those agents follow a previously established coordinated interaction based on execution of their own individual policies π_i. None of the agents should have an incentive to terminate outside the joint termination condition because once agents have decided to pursue the "subtask" represented by the option (and do not change their mind), the fact that the associated policies form a Nash equilibrium strategy should remove any incentive to deviate from this strategy. Following this reasoning, agents terminate execution of a multiagent option either (a) when that option reaches a probabilistic termination state or (b) when it is detected that the other agents are not participating in the selected option. This detection happens immediately after the first action is taken within an option, as the current implementation is fully observable. Other termination schemes for agent non-selection or early exit from an option choice will be discussed in Chapter 5.

The overall process individual agents use to select and execute multiagent options is illustrated in Figure 4.2. Beginning with the individual agent's receipt of observations, the agent will update the value for the most recently selected primitive action (here using NashQ, but this approach does not require a specific update process). If the agent is participating in a multiagent option, at this point, the agent will assess whether or not the agent would like to continue to execute this option. This will be based on the option termination criteria τ, however, the agent may also consider other factors such as the participation of other players in the option when determining to terminate. If the option has terminated, the agent updates the value of the multiagent option as described in Equation (4.1). If the option has not terminated, the agent selects the next primitive action to execute based on the agent's policy π_i within the option. If the option has terminated, or the agent was not participating in an option to begin with, the agent will select the next option (primitive or multiagent) to execute per the current selection approach (in NashQ, via a stage game comprised of the current Q-values for the state of the agent, if not exploring). If the option selected is not primitive, a primitive action will be selected for the next execution step. Once all participating

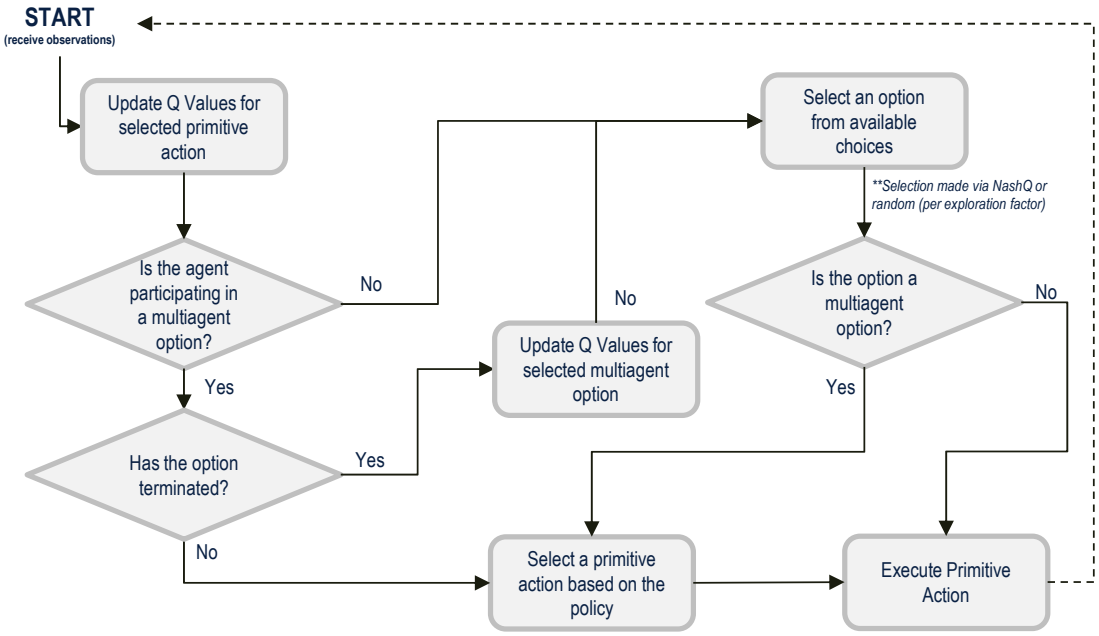

Figure 4.2 Agent Decision Process for Selecting and Updating Multiagent Options

Agents select primitive actions to execute based on the policies described by multiagent options. Upon termination of those options, they update the values of both the options themselves and the primitives that comprise the options.

players have selected their actions, those actions are taken, new observations are received, and the process starts anew.

The following sections will describe in detail the option selection approach and the value updates for multiagent options.

4.2 Selecting a Multiagent Option

At each decision epoch, the players use NashQ to determine the Nash equilibrium action they will play for the next cycle. This selected action could be a primitive action or a multiagent option. If both players choose to play the same multiagent option, they are effectively choosing to collaborate for the duration of that option. For the purposes of this work, Nash equilibria are identified by the GAMBIT game theory tool suite [41]. Further details about the configuration of GAMBIT can be found in Appendix A. Since the option is selected game-theoretically, the stage game used to select the option is illustrated in Figure 4.3. More formally, the game is described as $G=\{N,O_s,U\}$ where

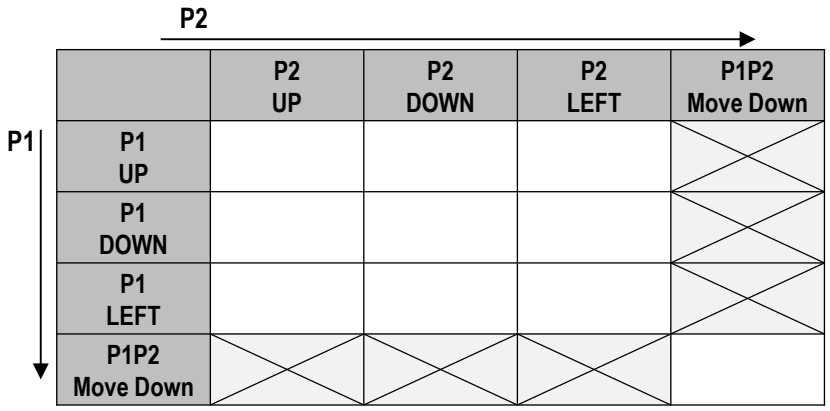

Figure 4.3 Stage Game with Multiagent Options

When using Nash equilibria to select options to execute, agents construct a stage game where the actions for each agent include all the options where the current state is an initiation state for that option.

N is the number of players, O_s is the set of options $o : s \in I_o$. {We define the set of primitive (single step) options to have $I = S$ and $\tau = S^+$ such that primitive options can begin and end in any state.} U defines the payoff function for each player.

4.3 Updating Values for Multiagent Options

Primitive actions are updated per the standard NashQ process as described in Equation (2.10) at every time step, but updating the value of the multiagent options is handled slightly differently. After an option o has completed, the value of its initiation state s_0 is updated per Equation (4.1).

$$Q_{(t+1)}(s_0, o) = [(1-\alpha) * Q_t(s_0, o)] + \sum_{i=0}^{\mu} \beta^u R_i + (\alpha * \beta^{(\mu+1)} * NashQ(s')) \qquad (4.1)$$

where μ represents the number of timesteps the option was active, and R_i is the reward received in intermediate state s_i which is a state traversed during option execution.

The assessment of whether or not all participating players have selected an option is handled through the full-observability of the system – all players know via observations exactly which (if any) multiagent option was selected by each independent agent and which primitive action was executed immediately after the first action is executed. This information can be used to terminate

the option choice if the other participating player has not explicitly selected the option as well. Other termination strategies are discussed and assessed in Chapter 5.

4.4 Summary

This chapter defined the concept of multiagent options, an extension to options that enables agents to work together on an ad hoc basis while maintaining their independent utility. In the following chapter, we describe experiments applying the multiagent option construct to a two player domain to assess learning time and termination approaches.

CHAPTER 5

Multiagent Options in a Two Player Domain

As a starting point in researching how agents collaborate with other agents while still maintaining self-interest, we have considered the case of two player games. Addressing the behaviors within a given set of agents is a critical first step in evaluating the behaviors across sets of agents. This analysis provides valuable insights into the process of selecting a multiagent option to execute across a distributed set of agents, the benefit of using multiagent options to reduce learning time, and termination approaches and their impacts across agents looking to leave a coalition. These techniques will be applied in later chapters to experiments where more agents are represented in the world.

5.1 Experimental Scenario

The experimental scenario considered is a 3x3 grid world with both competitive and cooperative tasks illustrated in Figure 5.1. The purpose of the game is for each player to locate one of two keys and bring it to their goal to open a player-specific door to a blocked region. Each key can only be picked up by a single agent when both agents occupy the same square as the key. To access the keys, the agents must cross a bridge square, which they can only enter together. If an agent attempts to enter the bridge square alone, that agent remains in place. If both agents attempt to pick up a key, neither agent gets the key. The game ends when at least one door is opened with the key, meaning that the agent and a key are in the goal square and a specific action is taken to open the door. Each agent is penalized for every time step, making the final score dependent on the time it takes to retrieve and use a key. This penalization as well as the difference in reward between the agent that picks up a key and the agent that only assists comprises the competitive aspect of the game. Agents receive a small bonus reward for holding a key and a slightly smaller reward for the other agent holding a key. There is a large reward associated with opening the door. All rewards aggregate at each time step.

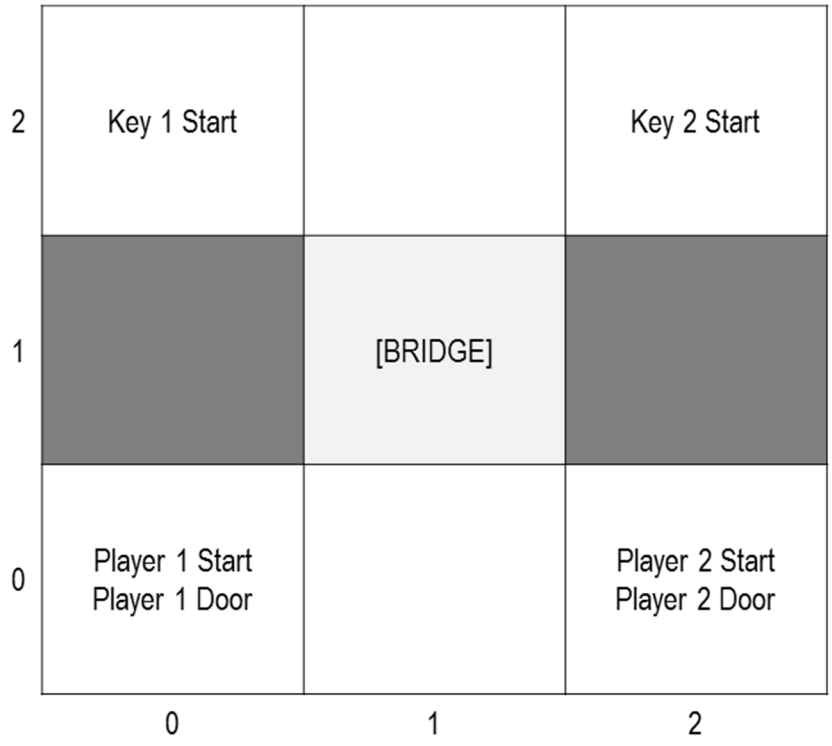

Figure 5.1 The Grid World

The grid world for the two player game contains cooperative and competitive tasks. Agents have independent utility, but need the aid of other agents to achieve their goals.

In this environment, the states are described by the positions of each player, the positions of each of the keys, and for each key, which (if any) agent possesses the key. These parameters describe the available state space $S: P_{1x} \times P_{1y} \times P_{2x} \times P_{2y} \times K_{1x} \times K_{1y} \times K_{2x} \times K_{2y} \times K_{1p} \times K_{2p}$; where P_1 and P_2 represent the players in the game, K_1 and K_2 represent the keys, $x: \{0,1,2\}$ and $y: \{0,1,2\}$ represent x and y positions in the grid world environment, and $p: \{NONE, P_1, P_2\}$ represents the possession of a given key. This results in 59,049 distinct states.

There are six available primitive actions *a = {up, down, left, right, pickup, drop}*. Overall, there are also six available multiagent options *o = {MoveUpToBridge, MoveDownToBridge, P_1GetsK$_1$, P_1GetsK$_2$, P_2GetsK$_1$, P_2GetsK$_2$}*. The MoveUp option coordinates agent behavior to get both agents

successfully to the bridge square from the bottom row of the grid world, while the MoveDown option does the same from the top row. The GetKey options coordinate among agents to find and retrieve keys. In each of these multiagent options the joint policy π represents a Nash equilibrium strategy for obtaining the associated task objective indicated in the name of the option. Both players $P = \{P_1, P_2\}$ must participate in these options. Rules defining the initiation states and the termination probabilities associated with these options are defined in Table 5.1. All states that meet the criteria indicated in Table 5.1 are members of the initiation set for that multiagent option.

A selected example path for two agents from the MoveUp option is illustrated in Figure 5.2. Each of the players has the ability to select a multiagent option in the associated initiation states for that option in addition to the primitive action choices (which are always available). No options are available to be selected when agents are split (one in the top row and one in the bottom row) or in the bridge square itself. The policies associated with the options represent Nash equilibrium strategies for their specific associated subtasks and, while they have been hand-coded in these experiments, could easily be learned policies via a sub-goal learning process.

Table 5.1 Option Initiation States and Termination Probabilities

This table illustrates the initiation states associated with the six two-player multiagent options, as well as the definition of termination probabilities.

Option	***Initiation State Definition***	***Termination Probability Definition***
Move Up To Bridge	$P_{1y} = 0$ & $P_{2y} = 0$ & ~($P_{1x} = 1$ & $P_{2x} = 1$)	1 if $P_{1x} = 1$; $P_{2x} = 1$; $P_{1y} = 1$; $P_{2y} = 1$ 0 otherwise
Move Down To Bridge	$P_{1y} = 0$ & $P_{2y} = 0$ & ~($P_{1x} = 1$ & $P_{2x} = 1$)	1 if $P_{1x} = 1$; $P_{2x} = 1$; $P_{1y} = 1$; $P_{2y} = 1$ 0 otherwise
Player 1 Gets Key 1	$P_{1y} = 2$ & $P_{2y} = 2$ & K_{1p} = NONE & $K_{2p} \neq P_1$	1 if $K_{1p} = P_1$ 0 otherwise
Player 1 Gets Key 2	$P_{1y} = 2$ & $P_{2y} = 2$ & $K_{1p} \neq P_1$ & K_{2p} = NONE	1 if $K_{1p} = P_1$ 0 otherwise
Player 2 Gets Key 1	$P_{1y} = 2$ & $P_{2y} = 2$ & K_{1p} = NONE & $K_{2p} \neq P_2$	1 if $K_{1p} = P_2$ 0 otherwise
Player 2 Gets Key 2	$P_{1y} = 2$ & $P_{2y} = 2$ & $K_{1p} \neq P_2$ & K_{2p} = NONE	1 if $K_{1p} = P_2$ 0 otherwise

The options themselves are each focused on an aspect of the game that requires coordination among the players to be successful, for example, both players being in the same square as a key with only one player attempting to pick the key up. This results in some sub-optimal individual behaviors for the agents when seen in relation to their final task rewards. For example, if an agent is already in the same square as a key, the other agent is not, and a "get key" option is jointly selected, the agent that is already positioned correctly will remain in that location (at individual cost) until the other agent arrives at that position.

If a player selects a multiagent option and the other player does not select the same option, the player will terminate option execution and reselect based on NashQ. This is enabled by the fully observable nature of the current implementation and allows our experiment to remain unconcerned with detecting if the other player has selected an option. If both players select the same multiagent option, that option executes until a common termination happens per the termination probabilities.

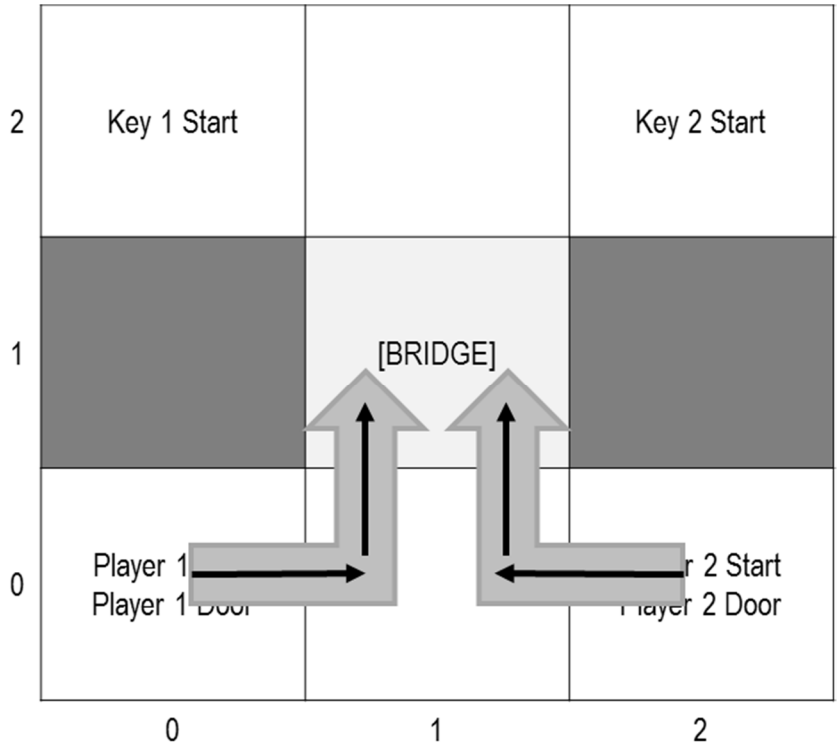

Figure 5.2 A Sample Path Taken By Two Agents Executing the "Move Up To Bridge" Option

When players both select the multiagent option, their policies allow for coordinated behavior.

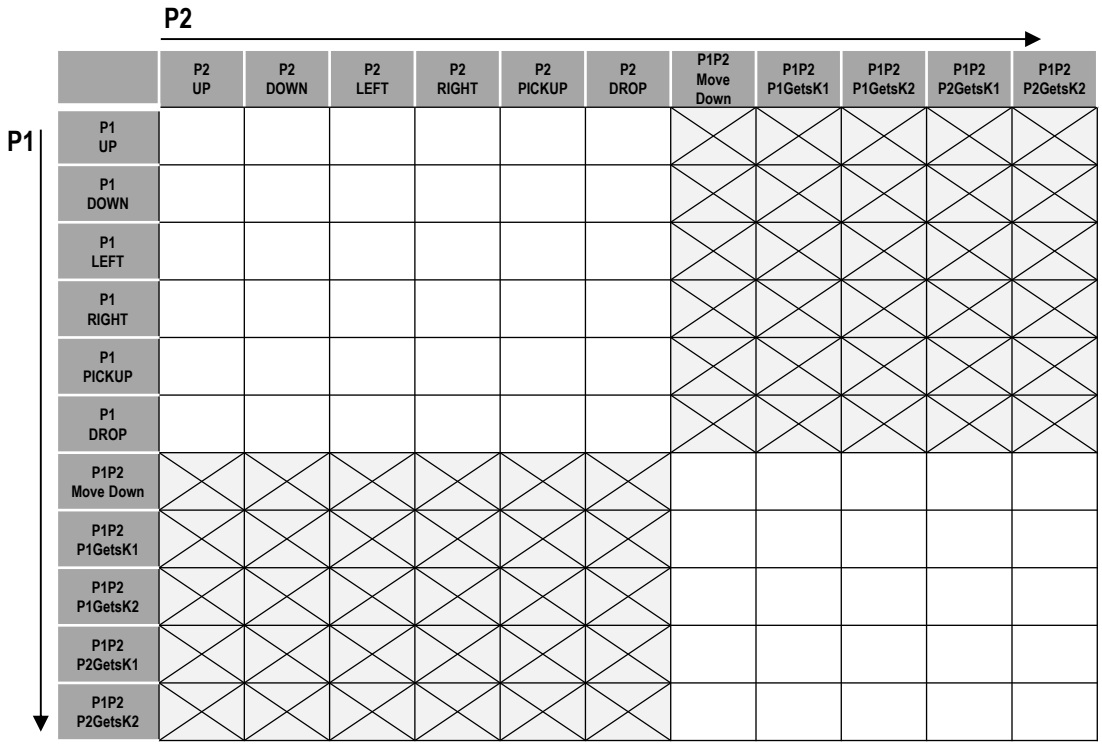

Figure 5.3 Player Game Table When Both Players Are In the Top Row of a Grid World and Are Not Holding Keys

The matrix game including multiagent options includes some values that would be in conflict with one another (shown as "X"s in this diagram).

To determine learning performance, these experiments were run for 10,000 episodes. Games were run with or without multiagent options available. Execution of an episode continued until a player opened a door or 500 primitive actions were executed. In the learning period, players would randomly select an option per an exponentially decaying rate (λ) of -0.0005, which would transition to a 1% exploration rate (ϵ) after the learning period was completed (episode 8400). Initial Q-values are set to 5, which is the reward when both players hold keys, without the movement penalty.

5.2 Option Selection

Selecting a multiagent option is a straightforward process. All possible options for P_1 are arranged as rows in the game, and all possible options for P_2 are arranged as columns as shown in Figure 5.3. Note that there are some unachievable states in this game (states where the same player

would be executing two options (one multiagent, one primitive) simultaneously). The update of these states is based on the termination approach used, and in the case where multiagent options terminate immediately if all players are not explicitly following the options, the player that selected the option never updates its value against any option but itself. Nash equilibria are found using the GAMBIT game theory tool suite [41].

5.3 Choosing Among Multiple Equilibrium

A challenge in a multiagent domain with independent decision makers is the selection of a consistent equilibrium across multiple agents. As discussed in Chapter 2, there can be multiple equilibria for a given game, and not all equilibria have the same utility. The utility of an equilibrium for a given player is defined in Equation (2.2). When agents select equilibria independently, it is critical that they consistently select the same equilibrium (otherwise they may not be selecting an equilibrium at all). To determine an appropriate scheme to select among multiple equilibria for the same game, four different equilibrium selection methodologies were assessed.

5.3.1 Select the First Equilibrium

A basic selection approach is to have all agents select the first equilibrium that is returned. This approach is straightforward to implement, and as long as all the players have the same game values, game structure, and the equilibrium selection process is deterministic in the order it generates results, will result in a consistent pick across agents. However, this approach gives no weight to the quality of the equilibrium strategy for any of the players.

5.3.2 Select the Equilibrium Based On Additively Combined Value

In order to insert some measure of equilibrium quality assessment, we can evaluate the value of the equilibrium quantitatively based on the maximum combined individual agent utility. The resulting equilibrium value is defined as in Equation (5.1). This mechanism provides a means to assess overall quality of equilibrium strategies, however, if a given equilibrium is very bad for a particular player and very good for another, this mechanism will return a neutral value, which may not be

representative of the actions that would be selected by a self-interested agent. This mechanism would be appropriate for purely cooperative tasks where the benefit to an individual agent would translate to an overall system benefit, but may not be as relevant in a self-interested, semi-cooperative domain.

.

$$v^{+}(e) = \sum_{i=1}^{n} U_{P_i}(e) \tag{5.1}$$

5.3.3 Select the Equilibrium Based On Multiplicatively Combined Value

Changing the combination mechanism to a multiplicative one (shown in Equation (5.2)) does handle the issue of equilibrium balance. Equilibrium strategies with low rewards adjust the value of the total combined score lower. One issue with combining utilities in this way is that utilities can be negative, resulting in adverse effects. To address this issue, a constant offset of at least the minimum possible utility is added to each player's equilibrium utility to ensure that all values are positive. Like the additively combined utility, this mechanism may be more appropriate for cooperative scenarios.

.

$$v^{\times}(e) = \prod_{i=1}^{n} \left[U_{P_i}(e) + c \right] \tag{5.2}$$

5.3.4 Select the Equilibrium Based On Maximum Risk

The loss associated with any strategy is the amount of utility a player would expect to lose (in the case of an equilibrium strategy, this is always a loss) if the other players continued to play the equilibrium strategy, but this player (and only this player) deviated. This is the "what's the worst that can happen" value. If we consider the support of a given strategy to be defined as $Z_{P_i}(e) = \{a \in A_i : p(a|e) > 0\}$, then we can define the set of unsupported actions for a given equilibrium to be $W_{P_i}(e) = \{a : a \in A_i, a \notin Z_{P_i}(e)\}$. We can then define the loss associated with any equilibrium as the maximum value that a player would lose if they deviated from the equilibrium strategy as in Equation (5.3).

$$L_{P_i}(e) = \max_{a \in W_{P_i}(e), b \in Z_{P_j \neq i}(e)} \sum p(b|e) * u_i(a,b) \tag{5.3}$$

Risk dominant equilibria, defined originally by Harsanyi [42], are equilibria that maximize the loss for all players as shown in Equation (5.4). This is the equilibrium that maximizes every agent's suffering if they choose to deviate.

$$v^R(e) = \prod_{P_i \in N} \left(U_{P_i}(e) - L_{P_i}(e) \right) \tag{5.4}$$

A risk dominant equilibrium is the one where players have the most to lose by selecting a non-equilibrium strategy, and therefore each agent has the greatest incentive to remain a participant in the equilibrium.

To assess the performance of each equilibrium strategy associated with the current 3x3 grid world scenario, ten different strategy selection methods were assessed. These 10 methods represented first and second choice selection options and are defined as follows: RISK ADDITIVE, RISK MULTIPLICATIVE, RISK FIRST, ADDITIVE RISK, ADDITIVE MULTIPLICATIVE, ADDITIVE FIRST, MULTIPLICATIVE ADDITIVE, MULTIPLICATIVE RISK, MULTIPLICATIVE FIRST, FIRST. Each method used one of the above strategies to calculate the value of the equilibrium, and, in the case of multiple equilibrium still having the same value, used a second, different technique to break the tie. If there were still multiple equilibrium with the same value after two selection criteria were applied, the players would select the first of these equilibria. Figure 5.4 shows the combined score of the participants and the percentage of the Q-table values reached for the four primary equilibrium selection strategies in this grid world (detail associated with the secondary selection strategies is not shown). As illustrated, all selection methods perform better than the approach of simply selecting the first equilibrium strategy, but the variance is high among all selection methods. This is due to the complexity of the overall grid world task and the amount of collaboration required between agents to reach the end goal and get the highest score. Risk dominance as a selection method was a high performer in both exploration and combined score. The multiplicative selection criteria also performed well, and was used in these experiments as the tie-breaking strategy.

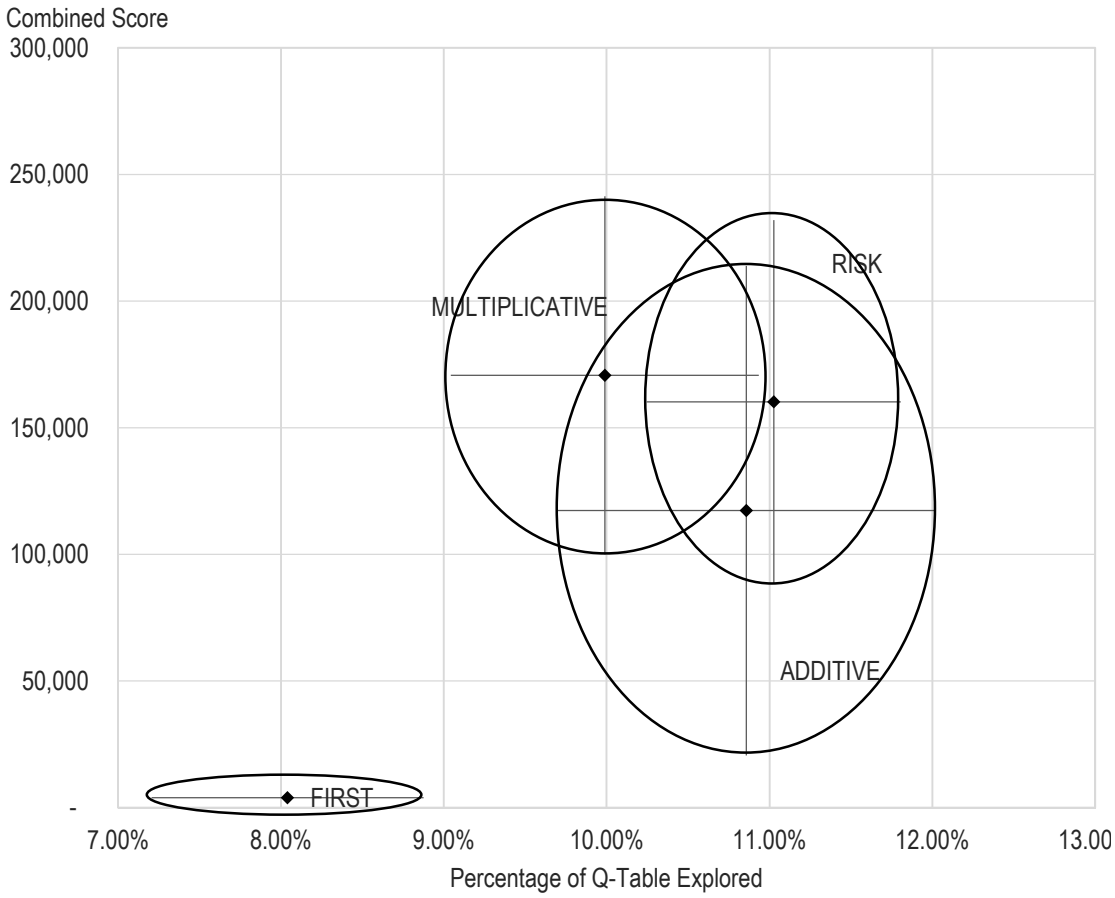

Figure 5.4 Performance of Different Equilibrium Selection Strategies

The above figure shows clearly the limited performance of the FIRST selection criterion. RISK, ADDITIVE, and MULTIPLICATIVE selection strategies all proved valuable in selecting successful equilibrium, with risk dominant selection strategies being a top performer.

5.4 Learning Time Impacts Due To Multiagent Options

Our experiments have shown that the ability to select multiagent options allows agents to move to distant parts of the solution space more quickly, and reduces learning time. Single agent options have proven to be an efficient mechanism to speed learning time for single agents by Provost, Kuipers, and Miikkulainen [10] who compared learning effectiveness in a robot-sensing environment between options-enabled and primitive action learners. They concluded that options-

enabled learners were able to rapidly identify critical areas of the state space because options moved high-value, distant rewards through the learning system more efficiently.

The score of both agents over the course of their learning is shown in Figure 5.5, with the multiagent-options-enabled agents represented with dark lines and the options-free agents with gray lines. Indications of the time during the experiments when the exploration rate reaches 0.1 and 0.01 are shown by the black diamonds. The agents with multiagent options available are able to learn the optimal policy of both agents opening their doors simultaneously. The agents without multiagent options available under the same learning time parameterization are only able to learn the sub-optimal policy of both agents retrieving keys under the same parameters. One of the key

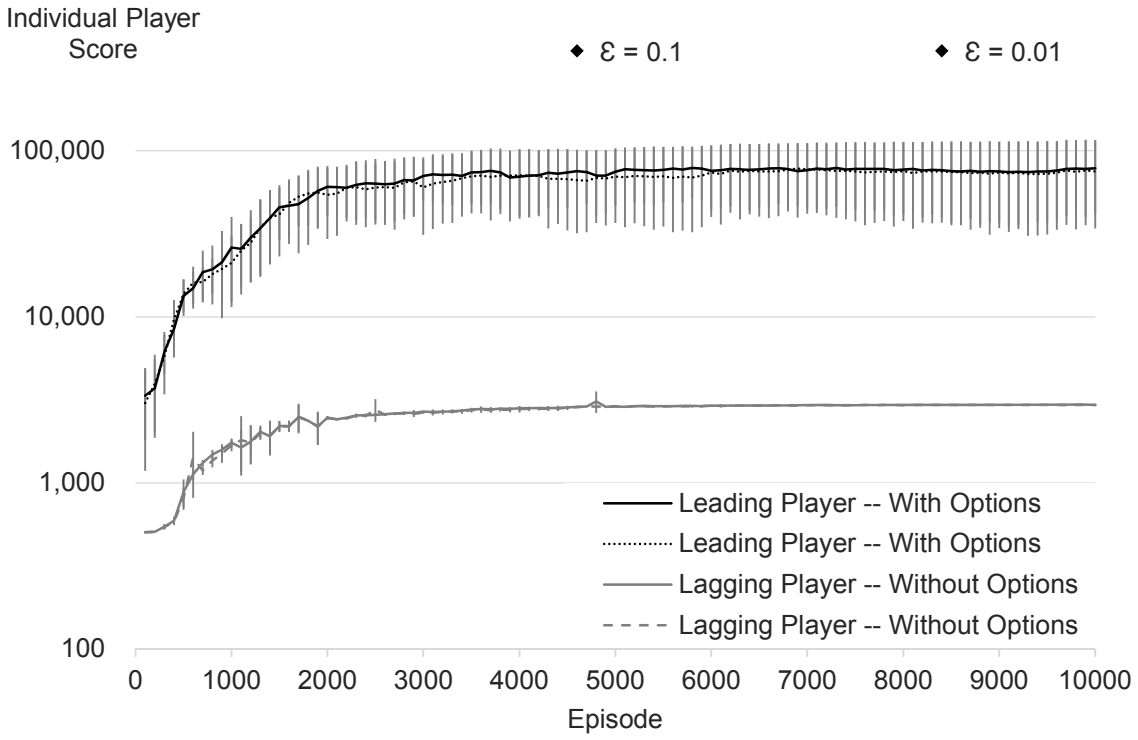

Figure 5.5 Comparison of the Score for Each Agent in a Two Agent Game with and Without Multi-Agent Options

*Each point represents the average score for 100 episodes across 25 runs with options and 5 runs without. The bars represent +/- one standard deviation. The exploration rate decreases exponentially from 1 at a rate of -0.0005*episode, to a minimum of 0.01. The black diamonds illustrate where the exploration rate (ϵ) = 0.1 and 0.01. Note these values are plotted logarithmically, and all values have been increased by a constant (1000) to ensure that they are depicted on this scale. After each run, the leading player was established as the player with the highest average score over the last 100 episodes.*

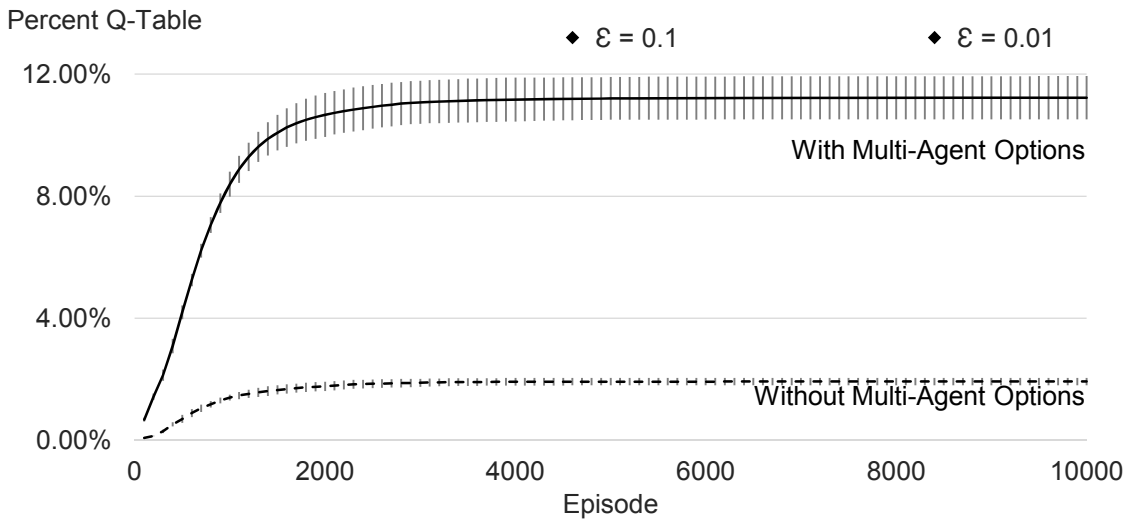

Figure 5.6 Percentage of Q-Table Explored With Multi-Agent Options and Without Multi-Agent Options

*Each point represents the total number of Q-table entries populated divided by the number of possible Q-table entries averaged over 100 episodes across 25 runs with options and 5 runs without. The bars represent +/- one standard deviation. The exploration rate decreases exponentially from 1 at a rate of -0.0005*episode, to a minimum of 0.01. The black diamonds illustrate where the exploration rate (ϵ) = 0.1 and 0.01.*

differences in our work to previous implementations of options in multiagent systems is that each agent makes decisions based on individual utility. As seen in these results, the agents with multiagent options learn the optimal policy together. Despite the fact that at least one of the non-options-enabled agents is able to reach the goal during the exploration period, those agents stabilize to a sub-optimal policy when the exploration period is concluded, because at least one agent does not have any learned incentive to cooperate beyond retrieving the keys.

As illustrated in Figure 5.6, multiagent options make more of the policy space (represented as a percentage of the Q-table explored) accessible to agents quickly. Agents are able to use multiagent options to leap forward to less easily accessible regions of the solution space and exploit the rewards available there. This result replicates the single agent domain findings of Provost, Kuipers, and Miikkulainen [10] in a multiagent domain.

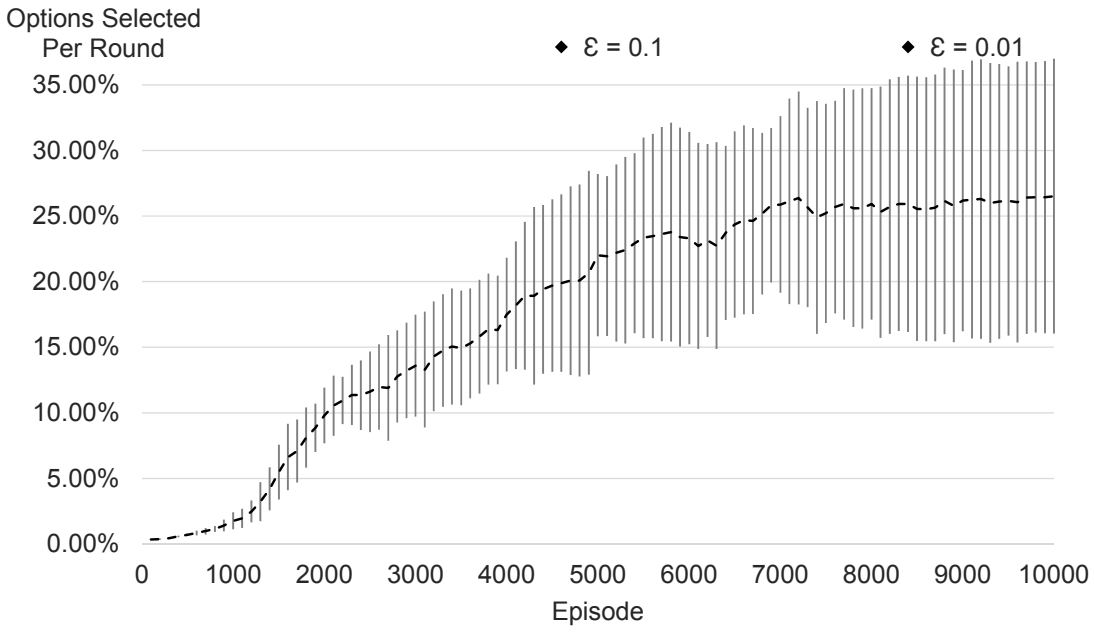

Figure 5.7 Options Selected Per Executed Round

*Each point represents the number of times a multiagent option was selected divided by the number of rounds in an episode averaged over 100 episodes across 5 runs. The bars represent +/- one standard deviation. The exploration rate decreases exponentially from 1 at a rate of -0.0005*episode, to a minimum of 0.01. The black diamonds illustrate where the exploration rate (ϵ) = 0.1 and 0.01.*

As shown in Figure 5.7, agents independently selected to collaborate via options a greater percentage of time as they learned. The large variance in option selection near the end of the runs is due to the fact that in some runs, after an initial learning period, the primitive actions within the option policies replaced the options themselves. Since options and their underlying actions initially resulted in the same Q-values, and since the update process was asymmetric (Q-values of primitive actions were updated as options executed, but options were not updated when primitive actions aligned with option-based policies executed), primitive actions have a slight learning advantage. In a few cases, once a policy was learned, the behavior was to “dissolve the option” and use the constituent actions instead. However, it is important to note that while the final policy no longer uses the options, it performs the same actions. The agent’s early use of options during learning is essential for the overall learning process, including the learning of the Q-values of the primitive actions used in the final policy.

5.5 Termination Analysis

The previous experiments terminated option selection when agents detected that the other participating player had not also explicitly selected the option (the first time step after the option was selected). To assess whether or not the joint behavior of selecting an option was critical to the success of multiagent options, several different termination strategies were considered and compared. Each player executes these termination strategies independently, though the strategy is consistent across both agents. This means that agents that selected an option simultaneously could terminate their execution of that option at different times, based on the termination strategy. The considered strategies would terminate the execution of the option:

- *When termination criteria was met.* This termination strategy would follow an option once it was selected until the termination criteria was met, regardless of the other agent's selection strategy. If a player was executing this termination criteria, and selected a multiplayer option while the other player did not, the player would execute the policy associated with this option until a termination criteria was satisfied, while the non-option-executing player would continue to select other actions.
- *When termination criteria was met or the option had executed for X timesteps.* This termination approach would follow an option until termination criteria was met or after the option had been executing for X timesteps, where $X = \{2,10,50,100\}$. If a player was executing this termination criteria, and selected a multiplayer option while the other player did not, that player would continue to execute per the option strategies until a termination criteria was reached, or until the option had executed for the specified amount of timesteps. Note that this termination criteria is subject to bias as the number of timesteps after which to terminate can be more successfully applied with some knowledge of the average successful option duration.
- *When termination criteria was met or with probability ρ.* This termination approach would follow an option until termination criteria was met, and would terminate each agent

independently in each timestep with probability $\rho = \{0.1, 0.2, 0.5\}$. If a player was executing this termination criteria, and selected a multiplayer option while the other player did not, that player would continue to execute per the option strategies until a termination criteria was reached, or until probabilistic termination.

- *When termination criteria was met or if all participating players were not explicitly following the policy.* This termination approach was the default approach used in the previous analyses, and would terminate options if both participating players did not explicitly select those options.
- *When termination criteria was met or if all participating players were not executing actions in the support of the option.* This termination approach assesses the behavior of the other participating players to determine if they are following the option implicitly (by selecting the primitive actions that comprise the policy), and if so, continues option execution. If another player deviates from the option strategy, the participating player terminates option execution.

An assessment of the final combined score and percentage of Q-table explored for the different selection methods described above is shown in Figure 5.8. Note that the implicit detection of whether or not players were executing option behaviors performed the best of all cases. Note also that the rapid termination of options (after 2 or 10 timesteps or with 20% or 50% probability) resulted in overall higher combined scores and high percentages of exploration, but also extremely high variations of performance. Performing worst was the approach where options are terminated solely per the termination criteria, or after long durations (100 steps).

Figure 5.9 compares the different termination strategies in terms of duration per round. The duration associated with termination of strategies by the termination criteria alone or by the number of timesteps executed is significantly higher (especially for options allowed to execute for long durations) in the early periods of high exploration, and then stabilizes to a similar relative time to other termination strategies.

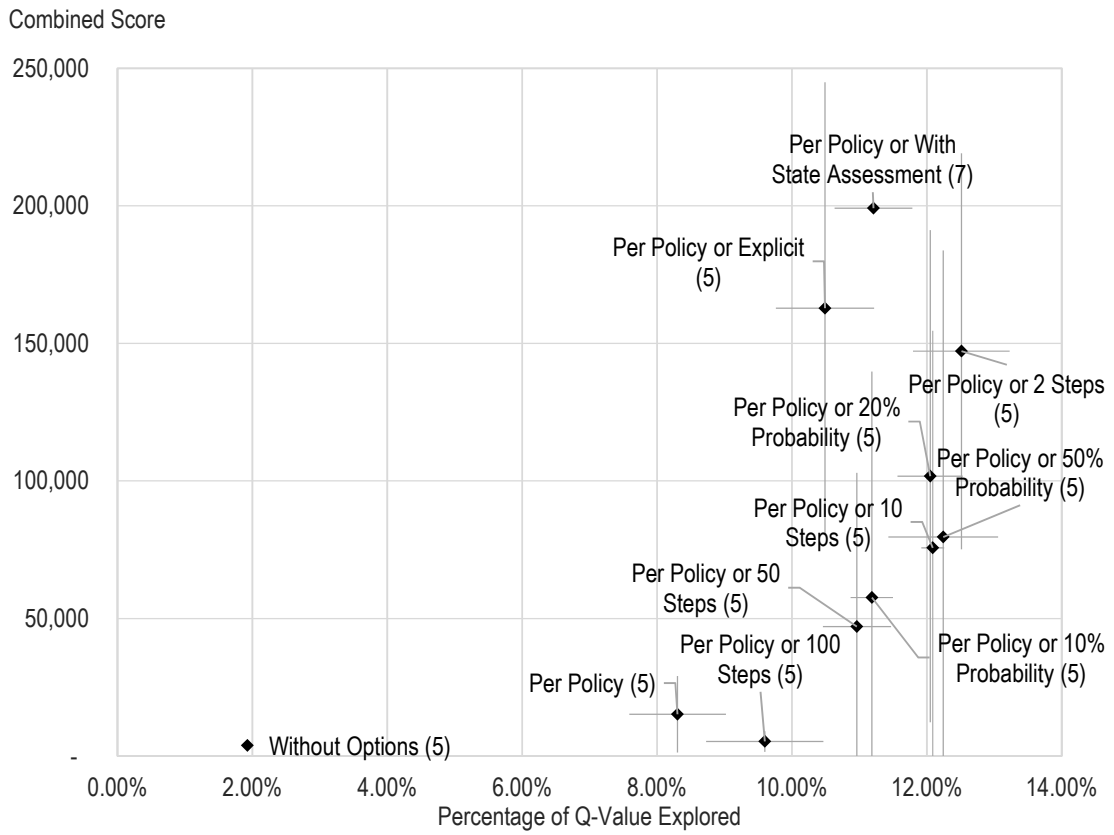

Figure 5.8 Termination Methods for Multiagent Options

The above figure shows the performance of termination criteria in terms of the final Q-value achieved (average over the last 100 episodes) and the combined final score for both agents (average over the last 100 episodes). The parenthetical numbers attached to each label are the number of runs completed with that termination strategy. The number parenthetically following each data label represents the number of experimental trials of each case.

5.6 Summary

The two player domain provided a rich source of preliminary information regarding option selection, learning time assessments, and termination approaches. Scaling these two-player multiagent options to environments with greater numbers of agents brings new challenges that are described in the upcoming chapters.

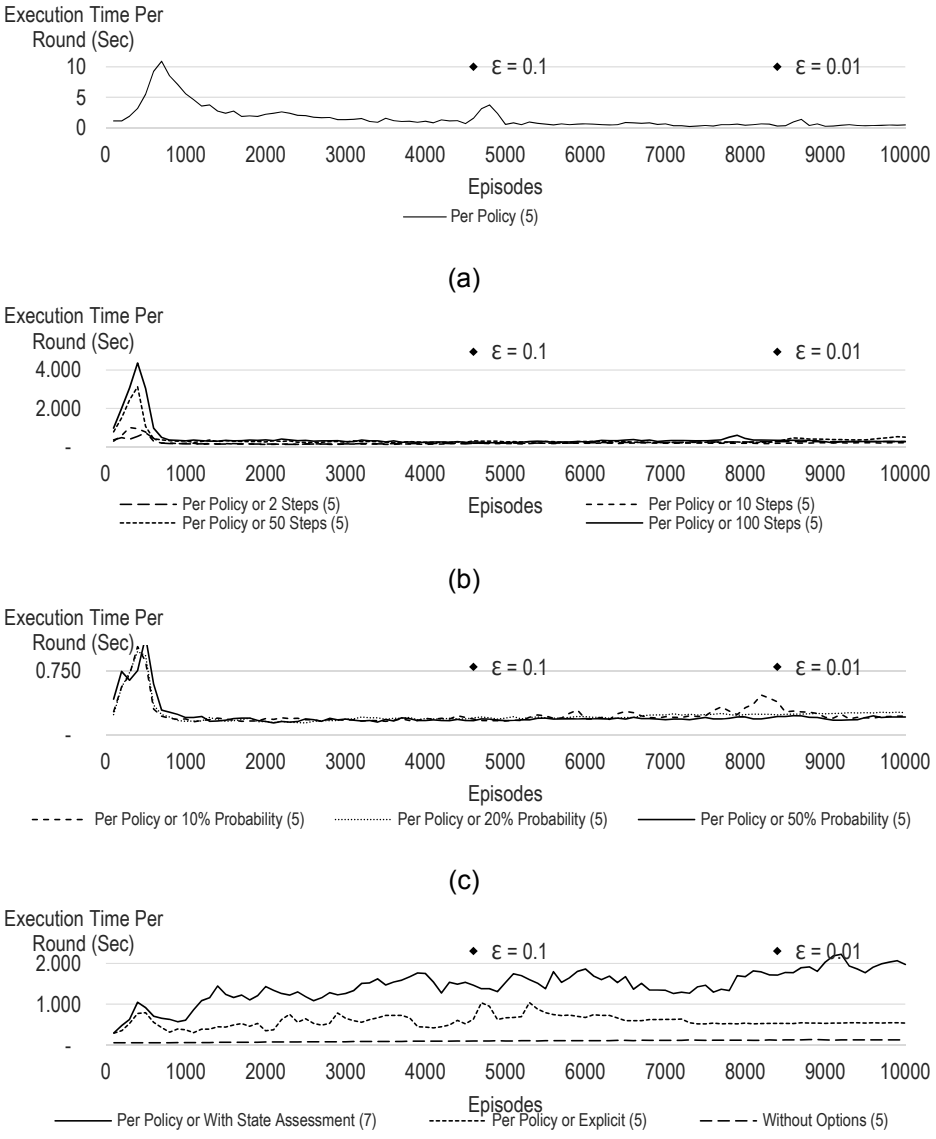

Figure 5.9 Execution Time Per Round for Termination Strategies

Each of these charts illustrates the execution time per round based on different termination strategies. Note that all of the vertical axes do not share a common maximum value. Chart (a) shows the values for execution based solely on the termination criteria. This was by far the most time-consuming termination strategy. Chart (b) shows the values for execution based on the number of timesteps executed, which was also significantly time-consuming, especially for the larger numbers of timesteps allowed. Chart (c) illustrates the values for termination based on a probability. These executed the most quickly, but did not have an impressive success rate. Chart (d) shows the values for termination based on assessment of if the other player is implicitly or explicitly following the option. These assessments took two to three times longer per round in later episodes, but the effective learning resulted in fewer numbers of rounds, so the overall duration of these runs was much shorter.

CHAPTER 6

Coalitions and Roles: Interactions between Agent-Sets

The previous chapters have focused on behaviors – multiagent options – which agents can select to enable coordination amongst a set of agents. In the two player domain there are really only two possible social choices – execute primitive actions as an independent agent (a coalition of one), or join forces with the only other agent to work cooperatively. The remainder of this dissertation focuses on how sets of agents interact with each other, and this chapter describes the mechanism used to represent a set of agents as a single entity.

6.1 Multiagent Options

Multiagent options (as defined in chapter 4), are extended time-step actions that facilitate coordination between multiple individual agents. To handle coalition formation, we extend the definition of multiagent options to include roles. Multiagent options are defined as a tuple $o = \{\Psi, \pi, \tau, I\}$ where Ψ is a set of roles that are required to execute the multiagent option, $\pi = \langle \pi_1, \pi_2, ... \pi_\Psi \rangle$ is a set of policies $\pi : S \times A \rightarrow [0,1]$ (one per role) mapping states to actions/options, $\tau : S \rightarrow [0,1]$ is a termination condition representing the probability of terminating the option, and $I \subseteq S$ is the initiation set which represents the set of states in which a multiagent option is valid to start.

6.1.1 Coalitions and Roles

We will model our environment via an MDP, which is a tuple $\{S, A, T, R\}$ where S is a set of states, A is a set of actions, $T : p(s, \vec{a}, s') \rightarrow [0,1]$ is the probability of transitioning to state s' from state s when action set $\vec{a}$ is selected, and $R : S \times A^N \rightarrow \mathbb{R}^N$ where $R(s, \vec{a}) = \vec{r}(s, \vec{a})$ is the reward received when an agent selects action a in state s. We will consider a world with N agents represented as a set P, each one with the ability to fulfill one or more roles.

For the purposes of this research, a coalition is a group of agents that have joined together to achieve a common goal or subgoal (expressed as a multiagent option), while maintaining independent individual utilities. We define a coalition as a set of players $c \subseteq P$. When players select a coalition to participate in, they also select an option to execute and a role to play in that multiagent option, which we refer to as a "bound option". In order for a coalition to form, all roles within the bound option must be assigned one and only one player, and no roles may be left unassigned. A bound option is defined as: $B = \{c, o, X\}$ where c is the coalition, o is the multiagent option that defines the coalition behavior, and $X : P \rightarrow \Psi$ is the assignment of specific players within the coalition to the option-defined roles. We define the set of bound options for a given player in a given state as ${}_pB_s = \{B : p \in c, s \in I_o\}$ where the player p is a member of the coalition and the state s is part of the initiation state of the option o.

6.1.2 Joining and Disbanding a Coalition

Agents game-theoretically select a multiagent option to execute, where the action space for a game spans all possible combinations of coalitions and multiagent options that can be executed by a given agent (defined in further detail in Section 6.1.6). If the selected option is fully populated (meaning all the roles are filled by one and only one agent who is capable to fill the role), the option executes until its termination criteria is met. If all roles are not filled, the option terminates immediately (after the first action choice) and new selections are made. In this way, coalitions of agents form and disband at the will and to the individual benefit of the agents participating in those behaviors.

6.1.3 Coalition Interactions with Other Coalitions

Coalitions, once formed, interact in the world with other coalitions, and thereby, the outcomes of their behaviors are influenced by the behaviors of those other coalitions around them. This concept is illustrated in Figure 6.1, where two agents (White and Black) will initially form a coalition to execute Option 1 together, while Gray executes per some other policy $\tilde{\pi}$, following which Black

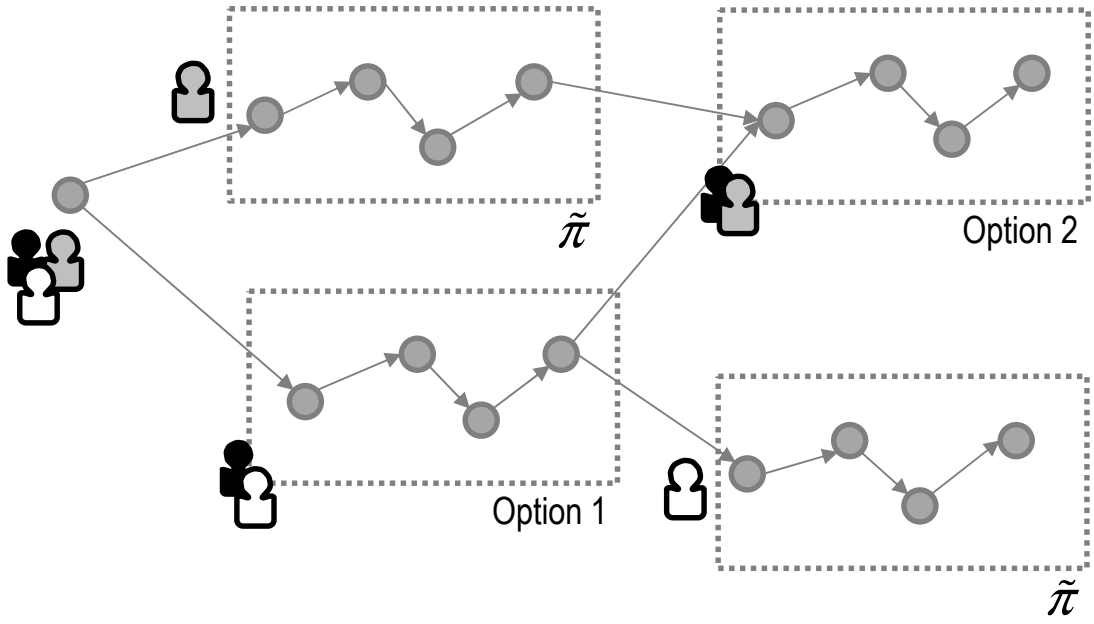

Figure 6.1 Conceptual Overview of a Partitioned Agent Set and Multiagent Option Execution.

In this scenario, three players (Black, White, and Gray) form shifting coalitions and alliances over time, with each coalition executing a multiagent option while the odd-man-out executes a best response policy $\tilde{\pi}$.

and Gray will align for Option 2 and White will execute independently (as a coalition of 1). Generally, this problem is represented in an stochastic game by a set P of N players, a set of states S (S^+ when termination states are included), and a set of primitive actions A. We extend the action space to include multiagent options, as discussed above.

To address the behaviors of external coalitions we also define $\tilde{\pi} : S \times A^m \rightarrow [0,1]$ as the policy for the m agents not in the given coalition, which at any given timestep t results in the selection of a set of primitive actions $\tilde{a}$.

The reward function for a given agent operating within an option can be considered as the probability that the coalition, subject to the other agents' policy $\tilde{\pi}$, will execute the option for a single timestep and not terminate, times the discounted reward for that step, plus the probability of transitioning to the next state and not terminating times the discounted reward for that state. Formally, this is defined by the recursive Equations (6.1) and (6.2). In the following equations, $\tilde{\pi}$

represents the policy of the other (non-coalition) agents and o represents the current selected coalition option.

$$ {}^{\tilde{\pi}}r^{o}(s,s',k+1) = \sum_{s''\in S}\left[{}^{\tilde{\pi}}p^{o}(s,s'') * (1-\tau(s'')) * \gamma * {}^{\tilde{\pi}}r^{o}(s'',s',k)\right] \tag{6.1} $$

$$ {}^{\tilde{\pi}}r^{o}(s,s',1) = {}^{\tilde{\pi}}p^{o}(s,s') * \tau(s') * \gamma * r \tag{6.2} $$

Using this, the reward for a given player when transitioning from s to s' can be defined as:

$$ {}^{\tilde{\pi}}r^{o}(s,s') = \sum_{k=1}^{\infty} {}^{\tilde{\pi}}r^{o}(s,s',k) \tag{6.3} $$

Discounted transition probabilities can be similarly defined as the likelihood that an option o terminates in a state s' after k steps after starting in state s_0 and thus as a function ${}^{\tilde{\pi}}f^{o}(s_0,s',k)$, where $\tilde{\pi}$ is the policy followed by all players not participating in the option. (Note that as in single agent options the discount factor is incorporated into this solution).

$$ {}^{\tilde{\pi}}f^{o}(s,s',k+1) = \sum_{s'\in S}\left[{}^{\tilde{\pi}}p^{o}(s,s'') * (1-\tau(s'')) * \gamma * {}^{\tilde{\pi}}f^{o}(s'',s',k)\right] \tag{6.4} $$

Where the first step is defined as:

$$ {}^{\tilde{\pi}}f^{o}(s,s',1) = {}^{\tilde{\pi}}p^{o}(s,s') * \tau(s') * \gamma \tag{6.5} $$

Using this, for all possible durations of a given option the likelihood (weighted by discount factor) that the agents end up in state s' when they initiate in state s is:

$$ {}^{\tilde{\pi}}f^{o}(s,s') = \sum_{k=1}^{\infty} {}^{\tilde{\pi}}f^{o}(s,s',k) \tag{6.6} $$

6.1.4 Learning Updates for Multiagent Options

We define the value of a state as the discounted sum of rewards gained in every state following that state, so the potential value of state s when selecting a multiagent option o that executes for k steps (where $\Upsilon(o,s,t)$ is the event of selecting option o in state s at time t) is shown in Equation (6.7). In the following equations μ represents a policy over multiagent options for coalition participants, and $\tilde{\mu}$ represents the policy over multiagent option for non-coalition participants.

$$^{\tilde{\mu}}V^{\mu}(s) = E\{r_t + \gamma r_{t+1} + \ldots + \gamma^{k-1} r_{t+k} + \lambda^k \, ^{\tilde{\mu}}V^{\mu}(s_{t+k}) \,|\, \Upsilon(\mu,s,t)\} \tag{6.7}$$

This, for all possible options in a given state translates into

$$^{\tilde{\mu}}V^{\mu}(s) = \sum_{o \in O_s, \tilde{\pi}' \in \tilde{O}_{s,o'}} \tilde{\mu}(s,o,\tilde{\pi}') \left[\, ^{\tilde{\pi}}r_s^o + \sum_{s' \in S} \, ^{\tilde{\pi}}f^o(s,s') \, ^{\tilde{\mu}}V^{\mu}(s') \right] \tag{6.8}$$

In Equation (6.8), $\mu(s,o)$ represents the probability that option o is selected in state s per policy μ (over multiagent options).

We can similarly extend these concepts to define the Q-value updates through option-over-multiagent-option μ, shown in Equations (6.9),(6.10), and (6.11).

$$^{\tilde{\mu}}Q^{\mu}(s,o,\tilde{\pi}) = E\{r_{t+1} + \ldots + \gamma^{k-1} r_{t+k} + \gamma^k \, ^{\tilde{\mu}}V^{\mu}(s_{t+k}) \,|\, \Upsilon(o,\tilde{\pi},s,t)\} \tag{6.9}$$

$$^{\tilde{\mu}}Q^{\mu}(s,o,\tilde{\pi}) = E\{r_{t+1} + \ldots + \gamma^{k-1} r_{t+k} + \gamma^k \sum_{o' \in O_s, \tilde{\pi}' \in \tilde{O}_{s',o'}} \tilde{\mu}(s_{t+k},o',\tilde{\pi}') \, ^{\tilde{\mu}}Q^{\mu}(s_{t+k},o',\tilde{\pi}') \,|\, \Upsilon(o,\tilde{\pi},s,t)\} \tag{6.10}$$

$$^{\tilde{\mu}}Q^{\mu}(s,o,\tilde{\pi}) = \, ^{\tilde{\pi}}r_s^o + \sum_{s'} \, ^{\tilde{\pi}}f^{o'}(s,s') \sum_{o' \in O_{s'}, \tilde{\pi}' \in \tilde{O}_{s',o'}} \tilde{\mu}(s',o',\tilde{\pi}') \, ^{\tilde{\mu}}Q^{\mu}(s',o',\tilde{\pi}') \tag{6.11}$$

Defining the rewards and transition probabilities in this manner allows us to consider the selection of multiagent options in a coalition framework as a single state transition in an MDP. We now apply these values in a stochastic game setting.

6.1.5 Partitioning the World

We fully model our environment as a N-player stochastic game, where a game is played in each state of an MDP. Each game is defined as a tuple $G = \{N, B_s, u\}$ where P is a set of N players, $B_s = {}_i B_s ... {}_{\mathrm{H}} B_s$ are the potential bound options available to sets of agents in state s, and $u = (u_i ... u_N)$, where $u_i : B_s \to \mathbb{R}$ represents the payoff value for agent i based on the bound option choices of all participants. Since we plan to evaluate the world in terms of a set of partitions of the agent set (restricting the number of coalitions that can simultaneously be active), we can also further define the payoff function in terms of that partitioning.

Given a vector of bound option choices picked by the agents $\vec{b}_s = (b_1, ..., b_N) \big| b_i \in B_{s,i}$, we extract the corresponding agent partition C_b, representing the set of coalitions involved in the resulting game, as follows: if all participants in a given coalition have selected the same bound option to execute (including multiagent option choice and role assignment), then the coalition is included in the coalition set. Otherwise, the primitive coalition is included.

We define valid H-partitions of the agent space, i.e. partitions of the agents into exactly H coalitions, as a coalition set $\hat{C}_{\mathrm{H}}$. For each coalition set in $\hat{C}_{\mathrm{H}}$, all players are part of one and only one coalition.

$$\hat{C}_H = \left\{ \{c_1, ..., c_{\mathrm{H}}\} \left| \begin{array}{l} \bigcap_{i:1..H} c_i = \{\varnothing\} \\ \bigcup_{i:1..H} c_i = P \\ c_i \neq \{\varnothing\} \end{array} \right. \right\} \tag{6.12}$$

A set of example player bound option selections, their corresponding C_b partition and an assessment of whether or not that partition is a member of $\hat{C}_2$ is presented in Table 6.1. As shown in the table, only bound option selections where all agents select the same behavior (option), the same organization (role mapping) and the same set of players (coalition) are considered valid

Table 6.1 Partitioning

This table provides some examples of partitioning based on bound option selection for a three player game.

Player 1 Selection	***Player 2 Selection***	***Player 3 Selection***	***Partition*** C_b	$C_b \in \widehat{C}_2$ ***?***
Move Up [RoleA: P1 RoleB:P2]	Move Up [RoleA: P1 RoleB:P2]	down	{P1P2, P3}	Yes
Move Up [RoleA: P1 RoleB:P2]	Move Down [RoleA: P2 RoleB:P3]	down	{P1, P2, P3}	No
Move Up [RoleA: P1 RoleB:P2]	Move Down [RoleA: P1 RoleB:P2]	down	{P1, P2, P3}	No
Move Up [RoleA: P1 RoleB:P2]	Move Up [RoleA: P2 RoleB:P1]	down	{P1, P2, P3}	No
up	down	down	{P1, P2, P3}	No

option choices. Any deviation in any one of those items will result in a partition that is not an element of $\hat{C}_{\mathrm{H}}$.

If action selection is restricted to only considering H participants, then only valid H-partitions are valid action choices in our stage game, and the payoff value for a given player, subject to the bound option choices of all players (which includes partitioning C_b) can be defined by the formula in Equation (6.13), where φ is a constant value less than the lowest possible value in the game. This formulation will also assign the value φ to otherwise valid (based on coalition alignment only) partitions where the coalition participants did not agree on role assignment or multiagent option.

$$u_i = \begin{cases} \varphi & \left| C_b \notin \hat{C}_{\mathrm{H}} \right. \\ {}^{\tilde{\mu}}Q^{\mu}(s,o,\tilde{\pi}) & \left| C_b \in \hat{C}_{\mathrm{H}} \right. \end{cases} \tag{6.13}$$

This implies that if any one agent deviates from their coalition, they will end up in an invalid partitioning of the system, as at least two agents must change (either via swapping roles or adjusting the composition of coalitions) to create a valid partitioning. This means that in the fully populated game, the only non-negative values in coalition "slices" are those where the agents have all agreed on the coalition to form, the option to play, and the assignment of players to roles in this option. Any deviation from this choice by any one of the coalition members will result in negative values.

At this point, we have structured a stochastic game over individual agents which reasons about partitions of the agent set, but we are still independently evaluating agent choice in a game space

of N agents. If we want to reason about coalitions of agents as a single entity for the scope of their existence, we must also combine the utilities of the agents within a coalition into a single payoff for the coalition in such a way that an equilibrium found using this coalition utility is also an equilibrium for the original agent utilities. Once the value of a coalition has been defined, the coalition can be represented as a single choice in an abstracted game where the number of players are based on the partitioning of the world.

There are several practical methods for combining individual utility into coalition utility. The most straightforward (and the one we will employ for the remainder of this chapter) is to maintain and update all the individual player's values, and combine the individual utilities into a bound-option centric game to make decisions on the actions to execute. This choice allows for a reduced stage game, which significantly simplifies bound option selection. The weakness of this approach is that the values of all individual players must still be maintained, and therefore this approach is limited in its ability to scale. Other approaches to calculating coalition utility are discussed in the following chapter.

6.1.6 Reduced Coalition Game Structure

In order to reason about coalitions of agents, we represent the world from the perspective of coalitions as a stochastic game, with the stage games associated with each state defined as $G = \{\mathrm{H}, B_s, u\}$, where H is the number of allowed partitions of the world (in following discussions and examples, for simplicity, we assume $\mathrm{H} = 2$), $B_s = B_{s,1},...,B_{s,\mathrm{H}}$ is the set of bound options available under each partition, and $u = (u_1,...,u_{\mathrm{H}})$ is the set of utilities (or payoffs) for each coalition. In regard to the structure of the game, we assume that the "decision-making" player p_i is always part of the first element of the coalition set (the "row player" in a two player game). This structure (referred to hereafter as the reduced game) is graphically depicted in Figure 6.2. Note that in this structure, all of the bound options available to player 1 are shown as row choices, and all of the bound options for disjoint partitions of the world are listed as column choices. This means that

Other Players

P1

	P3	P3	P2	P2	P2P3 P2: RoleA	P2P3 P2: Role B
P1P2 P1: Role A	u_{P1P2}, u_{P3}	u_{P1P2}, u_{P3}	φ, φ	φ, φ	φ, φ	φ, φ
P1P2 P1: Role B	u_{P1P2}, u_{P3}	u_{P1P2}, u_{P3}	φ, φ	φ, φ	φ, φ	φ, φ
P1P3 P1: Role A	φ, φ	φ, φ	u_{P1P3}, u_{P2}	u_{P1P3}, u_{P2}	φ, φ	φ, φ
P1P3 P1: Role B	φ, φ	φ, φ	u_{P1P3}, u_{P2}	u_{P1P3}, u_{P2}	φ, φ	φ, φ
P1	φ, φ	φ, φ	φ, φ	φ, φ	u_{P2P3}, u_{P1}	u_{P2P3}, u_{P1}
P1	φ, φ	φ, φ	φ, φ	φ, φ	u_{P2P3}, u_{P1}	u_{P2P3}, u_{P1}

Figure 6.2 Overview of a Reduced Format Stage Game.

Payoffs in the reduced format game are determined by whether or not the partition of the world is valid, and by the utilities of the bound options. Invalid partitions have a payoff to both players of φ, and valid partitions generate their payoff based on the utilities of the members of the coalition.

player 1 is simultaneously deciding what organization to be a part of, as well as what role to play in that organization's behavior. Note also that there are multiple invalid partitions of the world (represented as gray cells) in this table. For a given state $s \in S$, each coalition has a payoff value based on the option choice that is equal to the Q-value for that coalition for that action set or φ if the combined set of coalitions is not a valid partitioning of the player space.

Since we here consider that the equilibrium is centrally picked, a number of ways to construct a utility for bound options can be used that result in consistent picks (i.e. in picks that are also equilibria for the individual agents). To calculate the utilities for coalitions in this reduced game, we apply here the formula in Equation (6.14).

$$u_c = \begin{cases} \sum_{n \in c} u_n & c \in C_b \\ \varphi & c \notin C_b \end{cases} \tag{6.14}$$

We do not need to concern ourselves with the different scales of these payoff functions (the utility could increase based on the number of participants within a coalition), as they are only ever compared to the utilities of other coalitions of the same number of participants. This is evident in a

study of Figure 6.2, where all the column payoffs will be of the same scale (coalition vs. primitives) within that column, which is also true for the row player payoffs.

The major practical difference between this view of the world and the multi-player view is that there exist neighbors in this view are not necessarily neighbors in the full, agent-based table. Conceptually, the assessment of equilibrium relies firmly on the premise of row/column neighbors, as equilibrium is defined as an action-choice change for a single player. In this restructured table, the change represented is by all players within a coalition simultaneously, either to a different coalition action, a different role mapping within the same coalition action, or both.

6.1.7 Correspondence to the Full (Individual) Game

We construct the reduced table to enable decision making across sets of coalitions. When we are able to make decisions based on the reduced table, we can significantly reduce the scope of the game-theoretic problem. To make decisions from this construct, however, several properties must hold:

Multiagent options must be equilibria internally. Should this not be the case, it is unlikely that individually motivated agents would continue to execute an option that is outside of their own self-interest. However, we do not need to calculate an equilibria over the N-player game, only assess that the available multiagent option is an equilibrium (which is significantly less difficult).

Equilibria (specifically correlated equilibria) found in the reduced game must also be equilibria in the individual (full) game. Ensuring that this condition holds means that choices made in the reduced table are valid when assessed within the larger table.

It is conceptually helpful to notice a few things about the reduced table as compared to the fully representative table before the formal discussion. Figure 6.3 shows a fully representative table for a system with three players, broken out into “slices” based on the third player’s actions (these slices would be z-axis stacks in the fully representative game). Note that the only populated values in each slice are the valid H-partitions of the agent space. This means that for this player, the only populated values are those where the partition is valid, and corresponding coalition players have

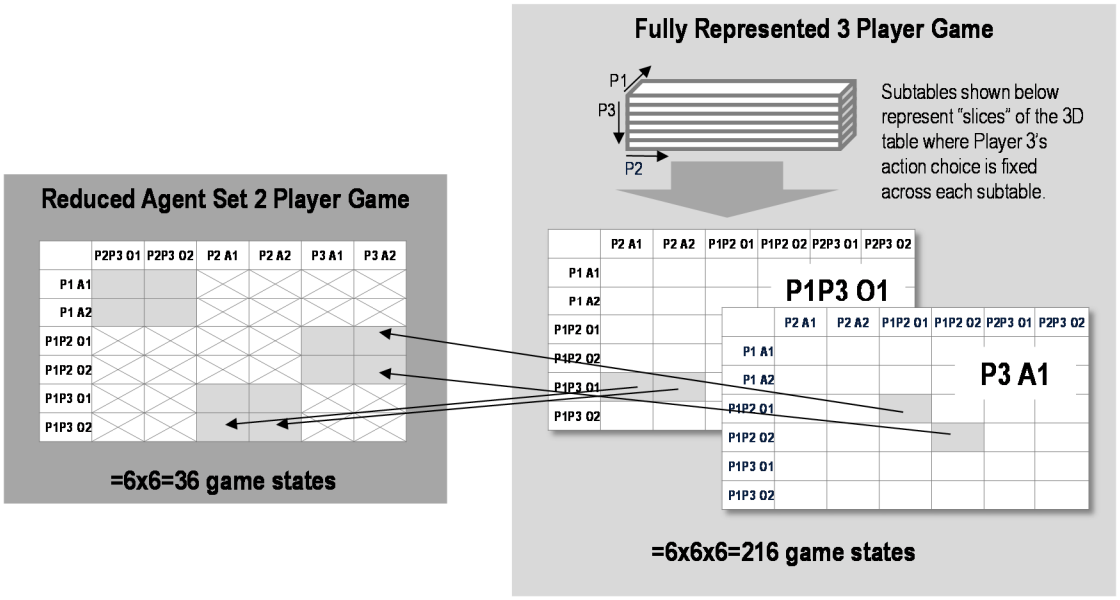

Figure 6.3 Mapping Of Game States between a Three-Player Fully Represented Game, and the Two Player Reduced Agent Set Game

While neighbors in the fully represented game are also neighbors in the reduced agent set game, coalitions also have additional neighbors, namely the other multiagent options available to the same coalition, regardless of role assignment.

selected equivalent bound options. All other choices would require a new partitioning of the world, or a new assignment of behavior internal to the coalition itself.

In the reduced game, however, coalitions (represented as a single game-action choice) can deviate as a group by selecting a new coalition action and a new role assignment using the same agents. Note that this also covers selecting a new action without changing the role assignment (assuming roles are consistent across options), or selecting a new role assignment without changing the option choice. Because this new game structure allows multiple agents (as a single coalition) to deviate simultaneously, it essentially creates "new neighbors" in the reduced game as compared to the fully represented game. Populated neighbors in the fully represented game are also neighbors in the reduced game.

Also of note is that the utility values are assigned to each individual player in the N-dimensional space. When these values are translated to the H-dimensional space, they must be combined into a single value function for the coalition executing the bound option. When translating from the fully

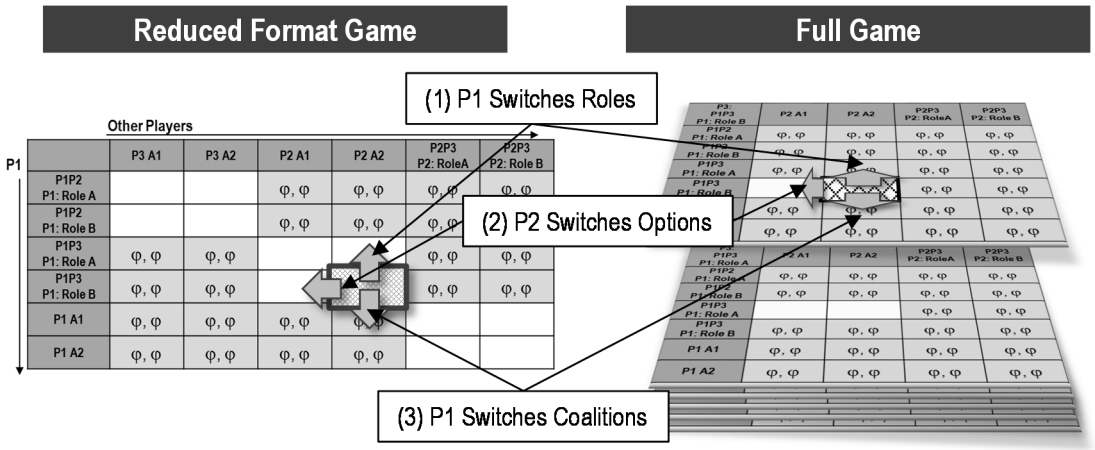

Figure 6.4 Summary of Potential Deviations in the Reduced Format Game

All multiagent coalition deviations in the reduced format game are not assessed in the equilibrium calculation for the full game. This is because, by definition, when a coalition contains more than one player, all players must deviate in order to switch locations. When coalitions contain only one player, those deviations are represented in the full table, and will be assessed as equilibria changes.

representative game to the reduced form game, we aggregate these individual utility values through the formula represented in Equation (6.14).

Claim. Correlated equilibria in the H-dimensional space are also correlated equilibria in the N-dimensional space.

Discussion.

If there is a correlated equilibrium in the reduced form game, deviating from that equilibrium takes one of three forms, illustrated in Figure 6.4.

(1) **A player can switch roles but remain in the same coalition with the same behavior.** This would involve, for example, switching from Move Up [Role A: P1 Role B: P2] to Move Up [Role A: P2 Role B: P1]. Note that the reduced format game cannot represent this switch for Player 1 without also representing the switch in role of Player 2. This type of deviation therefore will be considered in the reduced format game, but not in the fully populated game (because it requires more than one agent to simultaneously switch). Therefore, any switch of this type in the fully populated game will either (a) not be included

in an assessment of equilibria because more than one player changed their action or (b) will result in an invalid partitioning, and therefore have a payoff of φ.

(2) **A player can switch options, but maintain the same coalition and role mapping.** When this is a primitive option (as it is illustrated in the picture) this is by definition, a switch in the behavior of a primitive coalition. Therefore, the utility of the coalition is the same between the reduced format table and the fully populated table. If, however, the switch is for a non-primitive coalition, then accomplishing it requires all the coalition agents to switch behavior simultaneously and therefore this switch will not be considered in the equilibrium assessment of the fully populated table. Any single agent switch will result in an invalid coalition set, and therefore have a payoff of φ.

(3) **A player can switch coalitions.** Again, like above, this requires at least two agents to switch their action to maintain the dimensionality of the coalition set. For example, if a player is in a two person coalition, and switches to a single person coalition, then the player left behind is either in an invalid coalition, or has also switched coalitions.

Proof (Sketch).

A correlated equilibrium $\Phi = \{v, \varpi, \sigma\}$ in the reduced game is subject to the equilibrium condition shown in Equation (6.15).

$$\sum_{d \in D} \omega(d) u_i(\sigma_1, ..., \sigma_i, ..., \sigma_{\mathrm{H}}) \geq \sum_{d \in D} \omega(d) u_i(\sigma_1, ..., \sigma_i', ..., \sigma_{\mathrm{H}}) \tag{6.15}$$

Here, $v = (v_1, ..., v_{\mathrm{H}})$ is a set of variables from domains $D = (D_1, ..., D_{\mathrm{H}})$, ω is a joint distribution over v, and $\sigma = (\sigma_1, ..., \sigma_{\mathrm{H}})$ maps the variable selection over D to the bound option set B.

The option selection per the equilibrium will result in a corresponding partitioning C_b (probabilistically under ω). If $C_b \in \hat{C}$, there is a value for each coalition in the fully populated table. Otherwise, the value is φ.

For a single player coalition, the coalition payoff and the single player payoff will be the same for all bound option choices available to that coalition (see Equation (6.14)). Because Φ is an equilibrium strategy, any deviation by the (single player) coalition to a different bound option (again, for the single player coalition) will result in a decrease in value. Should the agent select a bound option that requires joining a coalition, an invalid re-partitioning of the coalition space will result (since only one agent can alter their actions, and joining a coalition by definition requires at least two agents to alter their decision), which will have a value of φ. Since φ is less than all possible payoffs by definition, for single player coalitions in the N-dimensional game, Φ is an equilibrium.

When considering multiplayer coalitions, in the N-dimensional game, if any one player within a coalition deviates from the bound option selection (either by selecting a new role, new multiagent option or new coalition), they create an invalid partition of the agent space, which results in a value of φ. Since φ is less than all possible utilities by definition, for all multiplayer coalitions, Φ is an equilibrium in the full table.

Therefore, since Φ is an equilibrium for all primitive and multiplayer coalitions in the fully populated table, any correlated equilibrium in the reduced format table is also a correlated equilibrium in the fully populated table.

6.1.8 Assumptions

In order to leverage this approach to structuring a game across agents, a few properties must be maintained. The first is that every state contains at least one valid bound option mapping that will allow the world to be partitioned into H pieces. Without this assumption holding true, there will be some states where there is no valid choice available for any of the agents. Similarly, if the "optimal" behavior for a set of agents requires a partitioning of more or less than H coalitions, this solution will never converge to that answer. Therefore multiagent option creation and application across the state space is critical.

The approach presented also assumes that a centralized agent will select the partitioning and provide that solution to the other agents to assess if the solution is an equilibrium or not. This allows

for the algorithm to focus on partitions of the system that are valid. If, however, the choice of coalition was distributed, both learning updates and convergence would need to be re-examined. This concept is discussed further in Chapter 8.

6.2 Experimental Analysis

To assess the viability of this approach, two different cases were executed in a grid world environment similar to that described in Chapter 5, and more fully described in Chapter 7. In the first case (traditional), agents maintained their individual utilities and a centralized agent selected actions from a three player table (with the incorporation of φ for all choices that would result in invalid agent partitions). The second case used the method described above to combine the individual agent utilities into a reduced format game, which was then used to select actions.

These experiments used Nash SARSA (described in section 2.1.5) as the learning technique to update Q-values. Experiments were run for 10,000 episodes. Execution of an episode continued until a player reached a winning state or 100 timesteps were executed. In the learning period, players would randomly select an option per an exponentially decaying rate (λ) of -0.0005, which would transition to a 1% exploration rate (ϵ) after the learning period was completed (episode 8400). Initial Q-values were set to 0.

6.2.1 Convergence

The convergence of these solutions is presented in Figure 6.5. The results have been truncated to 6,000 episodes due to extended learning times for the three player solution. Twelve experiments were executed for the traditional case and the reduced format game. While the reduced format game reaches the optimal collaborative policy consistently, the traditional approach does not.

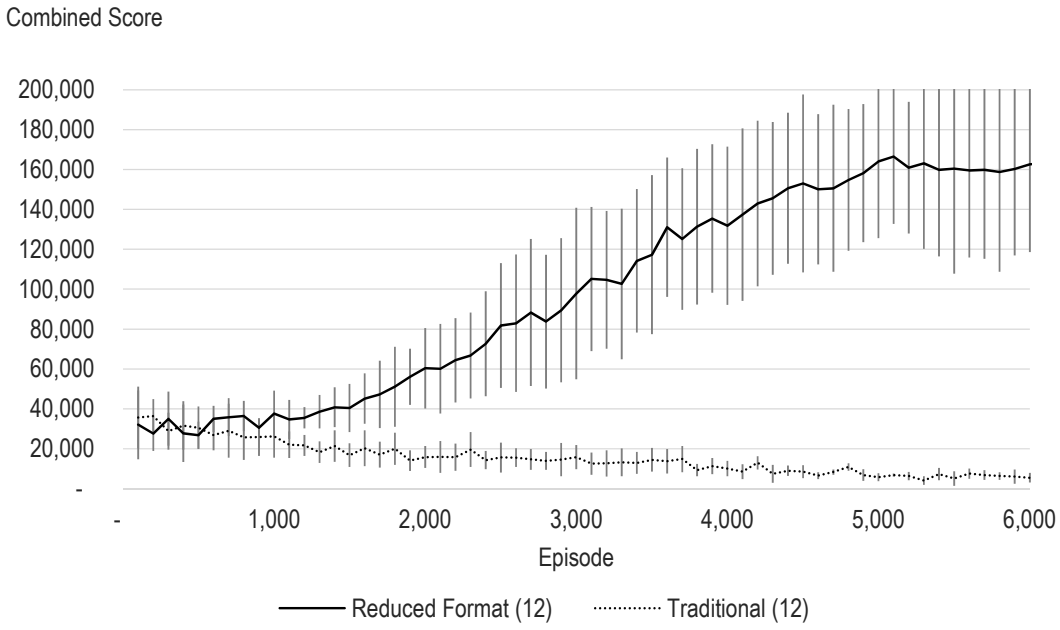

Figure 6.5 Learning Performance

This graphic displays the first 6000 episodes of the experiment. Twelve runs were executed for each of the traditional and reduced format cases. At this point, the reduced format games consistently arrive at the optimal solution, while the traditional three-player game does not find an optimal solution.

6.2.2 Computation Time

The required time to execute an episode was logged for each run. This time includes the period from game start to game end (the duration of an episode). Since one of the players returning home with the flag would result in the game ending, games can have varying numbers of rounds. Figure 6.6 presents the time required per round for the traditional versus the reduced format case, over 12 runs each. The display of this data is truncated at 6,000 episodes due to the long computation times of the traditional case.

The significant computation time difference between the two approaches is predominantly due to the time required to calculate an equilibrium solution. Stage games in this particular reduced format game range in size from 140 to 440 game states. Stage games in the fully populated game range in size from 1,000 to 10,648 game states. For this reason, many of the executed rounds have one or more games that time out the equilibrium solving algorithm (see Appendix A for further details), which adds an additional 10 seconds to the calculation. Although the structure of both types of stage games results in significant numbers of dominated strategies, the computation time for the much larger fully populated game is reflected in these results.

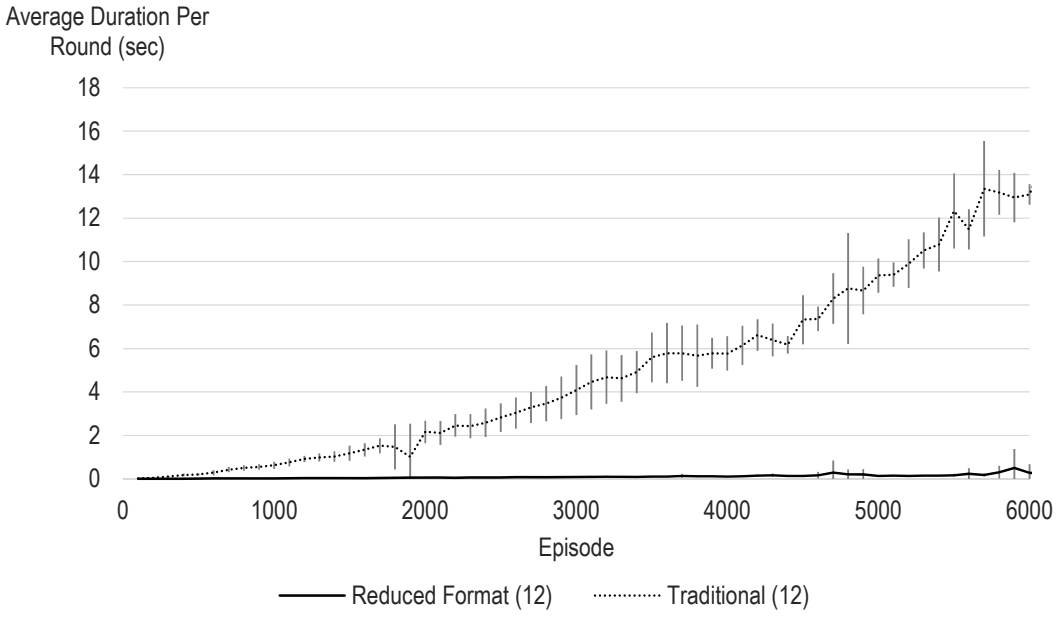

Figure 6.6 Total Equilibria in the Reduced Format Game Which Are Not Equilibria in the Fully Populated Game

As exploration decreases, more games are required to be played to determine the next action. Since the traditional approach uses the three player game to determine the next action in each state – some of which have over 10,000 game states, much more time is required to reach an overall solution.

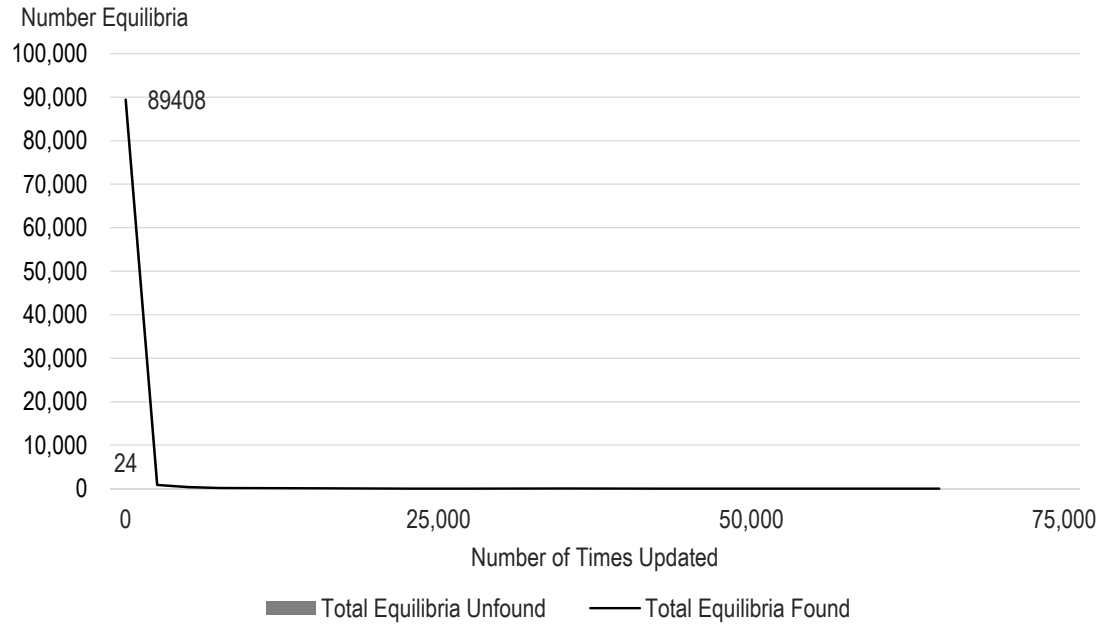

Figure 6.7 Total Equilibria in the Reduced Format Game Which Are Not Equilibria in the Fully Populated Game

Equilibria for stage games in nearly 32,000 states were calculated and assessed against the correlated fully populated games. Of the nearly 92,000 equilibria found, 24 of them were not valid in the corresponding fully populated game. The number "24" in this graph is a data label for a bar, however, the relative size of these values is such that the bar is not visible at this scale.

6.2.3 Equilibria

Figure 6.7 shows the equilibria that were selected in the reduced format game which were also not equilibria in the fully populated game. Of the nearly 92,000 equilibria found, 24 of them were not valid in the corresponding fully populated game, which makes for a success rate of 99.97%. The average deviation over these 24 equilibria was .31, which could be due to numerical computation issues or due to some portions of the solution not being completely converged.

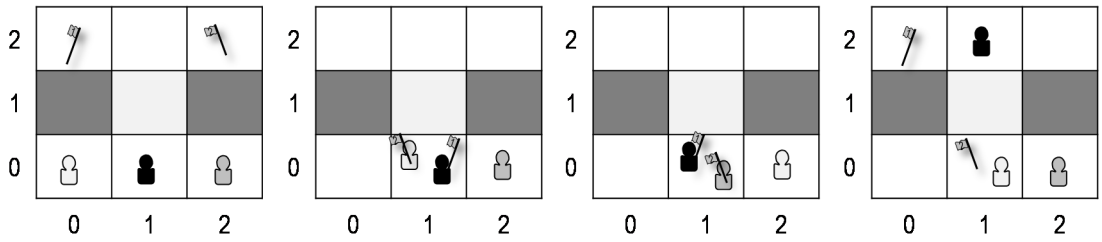

Figure 6.8 States Where the Equilibria in the Reduced Format Game Differed From the Equilibria in the Fully Populated Game

Of the ~30,000 states evaluated, equilibria for reduced format games in only four of those states did not match equilibria in the fully populated games. Graphical depictions of those states are represented above. They include the start state (far left), two states that are two steps away from at least one player (maybe both) winning the game, and a state where one player is one step away from winning the game.

The states in which these equilibria did not match is also interesting. These states are graphically depicted in Figure 6.8. They include the start state, and three states that are each within two steps of the game terminating because one or more players has won. In these four states, on average 82% of the equilibria returned from the reduced format game were also equilibria in the fully populated game.

6.3 Summary

This chapter has defined a new mechanism for construction of a stochastic game for multi-agent systems, whereby agents reason about a socially abstracted world state (us-vs-them). This mechanism has been shown to preserve equilibrium properties when assessed against a game of individual actors with equivalent action spaces and payoffs, resulting in a significantly reduced game state space and thereby more efficient game solution times. The following chapters describe experimental results of employing this model as well as extending the concept to distributed action selection.

CHAPTER 7

Evaluating Value and Reward with Multiagent Options

The previous chapter described a mechanism to select an action based on coalitions and bound options. However, this mechanism required that Q-tables representing the value for each individual player be maintained and updated, and then those values be combined in the reduced game structure to determine coalition actions. The previous chapter showed that equilibria calculated based on the combination of those independent player utilities would also be equilibria in games calculated without that combination. There are, however, significant advantages in terms of scale to maintaining values for a coalition (as opposed to individual values).

7.1 Motivation

When we use the reduced game to calculate actions, the values for any coalition action are combined based on values maintained in Q-tables associated with individual players per Equation (6.11) defined in the previous chapter. In order to maintain these values on a coalition basis, thereby removing the need to maintain individual values, two major issues must be addressed. The first of these issues is the issue of reward – namely how do we combine two individual rewards in such a

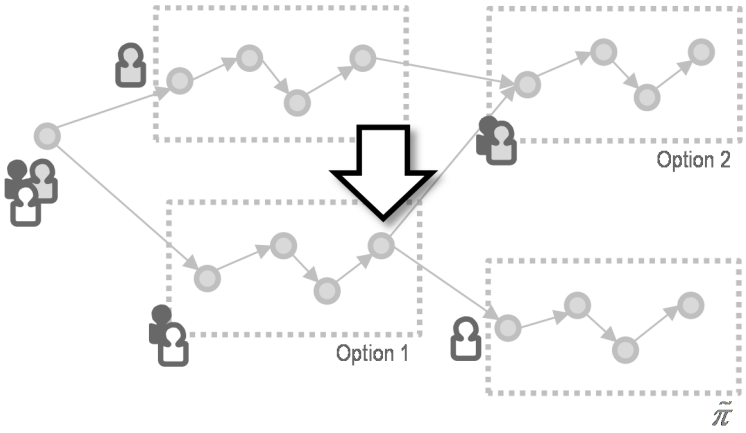

Figure 7.1 Updating the Future Values Of Coalitions

When a coalition terminates, the constituent players form new coalitions. This means that the future expected value of any given coalition must depend on the future states of its participants and must be computed across Q-tables.

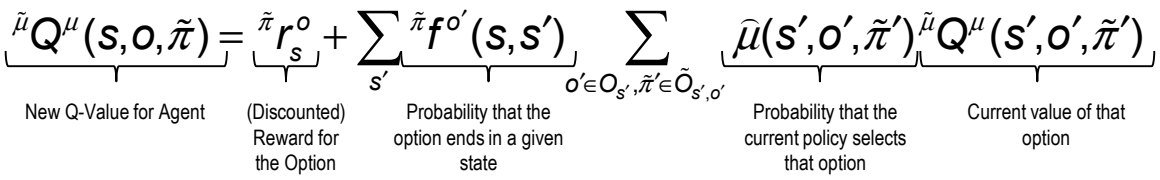

Figure 7.2 Annotated Value Update Equation for Individuals

When updating the Q-values for individuals, it is helpful to consider that value as comprised of two pieces – reward for what was immediately achieved (in this case, the last executed multiagent option), and the value of ending up in the current state.

way that the gradient of those rewards (representing the equilibrium) is maintained. The second issue is illustrated in Figure 7.1. At the termination state of any coalition, shown as "Option 1", the future value for the participants depends on their formation of (potentially different, as shown) coalitions. Therefore the Q-values for the White-Black coalition in Option 1 must be a combination of the value that Black gets from participating in Option 2 (in addition to any other probabilistically possible future next steps) as well as the value White gets from their future next steps. This did not need to be considered when constructing the reduced form game from individual Q-tables, as the updates of learning values based on observations occurred in the individual tables as is standard.

7.2 Combining Values and Rewards

Figure 7.2 shows the Q-value update equation presented in the previous chapter. This update equation sets the new Q-value for individual players to be equal to the reward achieved by that player over the course of the option, plus the expected value of actions taken in future states. This learning update happens when an option terminates, and for all intermediate steps within the option. We generate the updated value in three steps: 1) we combine the rewards for the agents participating in the option; 2) for each possible next state, we calculate the amount of individual utility expected for each agent based on the future coalition's Q-tables; 3) we combine the individual utilities into a coalition utility. The newly calculated coalition rewards and utilities are used in the traditional Q-value update equation (6.11) to update and maintain coalition Q-values. We would like these utilities to correspond to the individually generated utilities, such that equilibria generated based on this approach correspond to the equilibria generated by the individual agents. In addition,

we also would like to not have to maintain the individual Q-tables, so an approach that relies solely on coalition Q-tables is preferred.

We define the combined coalition reward value as ${}^{\tilde{\pi}}_{C}r^{o}_{s}$, which is the reward for coalition C generated over the course of option o and non-coalition policy $\tilde{\pi}$ beginning in state s. When allocating the expected future values of the coalition to the individual coalition participants, an allocation factor $\phi : \Psi \rightarrow [0,1]$ is applied.

Therefore, the allocation of expected future value to individual agents is as shown in Equation (7.1), where the value v allocated to a given agent $p \in C$ in future state s' is equal to the sum over all possible future options $o' \in O_{s'}$ in combination with non-coalition policy $\tilde{\pi} \in \tilde{O}_{s',o'}$ of the probability (over options) that option o' is selected in state s' times the Q-value for coalition C' in state s' times the allocation of value to player p when participating in role ψ_p in option o' as a part of coalition C'.

$$ {}_{p}v_{s'} = \sum_{o' \in O_{s'}, \tilde{\pi}' \in \tilde{O}_{s',o'}} \left[\mu(s',o',\tilde{\pi}') \, {}^{\tilde{\mu}}_{C'}Q^{\mu}(s',o',\tilde{\pi}') \phi(o',\Psi_p) \right] \tag{7.1} $$

The allocation of future value to individual agents can now be combined into a single expected future value for the coalition ${}_{C}v_{s'} : \mathbb{R}^{|C|} \rightarrow \mathbb{R}$.

This results in the previously defined Q-value update Equation (6.11) being restated for coalitions as defined in Equation (7.2).

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'} \left[{}^{\tilde{\pi}}f^{o'}(s,s') \, {}_{C}v_{s'} \right] \tag{7.2} $$

Five different approaches for generating ${}^{\tilde{\pi}}_{C}r^{o}_{s}$ and defining ϕ and ${}_{C}v_{s'}$ were assessed, and are presented in the following sections.

7.2.1 Sum

Combining the rewards for coalition members as the sum of the individual participants represents the coalition reward as the total value achieved by coalition members. The approach to do so is shown in Equation (7.3).

$$ {}^{\tilde{\pi}}_{C}r^{o}_{s} = \sum_{P \in C} {}^{\tilde{\pi}}r^{o}_{s} \tag{7.3} $$

When allocating the expected future values of the coalition to the individual participants, we define ϕ to be an allocated percentage of the overall option reward. This is defined once for a given option. When combining the individual values, ${}_{C}v_{s'}$ is defined as shown in Equation (7.4).

$$ {}_{C}v_{s'} = \sum_{P \in C} {}_{p}v_{s'} \tag{7.4} $$

Equation (7.5), shows that maintaining the sum of the individual Q-values is equivalent to maintaining the values themselves. However, this approach may not be the most fair, in that large rewards for an individual player could dominate the smaller (possibly negative) rewards for other coalition members.

$$
\begin{aligned}
{}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) &= {}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \\
{}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) &= \sum_{P \in C} {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) \\
{}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) &= \sum_{P \in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right] \\
{}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) &= \sum_{P \in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} \right] + \sum_{P \in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right] \\
{}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) &= {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{P \in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right] \\
{}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) &= {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')\sum_{P \in C}\left[{}_{P}v_{s'} \right] \right]
\end{aligned}
\tag{7.5}
$$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{C}v_{s'} \right] $$

7.2.2 Normalized Sum

Representing coalition rewards as the sum of coalition rewards, normalized to [0,1], results in the coalition reward represented as in Equation (7.6).

$$ {}^{\tilde{\pi}}_{C}r^{o}_{s} = \frac{\sum_{P\in C}\left[{}^{\tilde{\pi}}r^{o}_{s} + abs(\delta) \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} \tag{7.6} $$

When allocating the expected future values of the coalition to the individual participants, we define ϕ to be an allocated percentage of the overall option reward. This is defined once for a given option. When combining the individual values, ${}_{C}v_{s'}$ is defined as shown in Equation (7.7).

$$ {}_{C}v_{s'} = \sum_{P\in C} {}_{p}v_{s'} \tag{7.7} $$

As described previously,

$$ {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \sum_{P\in C} {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P\in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P\in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} + \frac{\sum_{P\in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} \tag{7.8} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \frac{\sum_{P\in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \frac{\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')\sum_{P\in C}\left[{}_{P}v_{s'} \right] \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')\frac{\sum_{P\in C}\left[{}_{P}v_{s'} \right]}{\sum_{P\in C}\left[abs(\eta - \delta) \right]} \right] $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{C}v_{s'} \right] $$

7.2.3 Product and Normalized Product

When considering combining coalition rewards as the product of individual coalition rewards where the value δ is a constant that represents the minimum reward achievable by a player. This scaling factor ensures that an individual reward will not be zero or negative.

$$ {}^{\tilde{\pi}}_{C}r^{o}_{s} = \prod_{P\in C}\left[{}^{\tilde{\pi}}r^{o}_{s} + abs(\delta) + 1 \right] \tag{7.9} $$

When allocating the expected future values of the coalition to the individual participants, we define ϕ to be an allocated percentage of the overall option reward. This is defined once for a given option. When combining the individual values, ${}_{C}v_{s'}$ is defined as shown in Equation (7.10).

$$ {}_{C}v_{s'} = \prod_{P\in C} {}_{p}v_{s'} \tag{7.10} $$

However, when seeking to combine these values into a coalition utility, (7.11) illustrates that representing the inner terms (reward times value) of this product become an issue. An option at this point is to maintain the individual rewards for players and combine them as previously discussed. However, for the purposes of this research, we will not consider product or normalized product as an evaluation mechanism.

$$ {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \prod_{P\in C} {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) \tag{7.11} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \prod_{P\in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right] $$

7.2.4 Average

Representing coalition rewards as the average of individual rewards provides a measure of fairness to the assessment. It also means that the values we are maintaining for the coalition do not

represent the total aggregate reward, but rather the average payout to each of the coalition participants.

$$ {}^{\tilde{\pi}}_{C}r^{o}_{s} = \frac{\sum_{P \in C} {}^{\tilde{\pi}}r^{o}_{s}}{|C|} \tag{7.12} $$

Because this reward is represented as an aggregate reward, we define $\phi = 1$. This means that when considering a future coalition, the agents expect to receive the average reward for coalitions forming in that state. When combining the individual values, ${}_{C}v_{s'}$ is defined as shown in Equation (7.13).

$$ {}_{C}v_{s'} = \frac{\sum_{P \in C} {}_{p}v_{s'}}{|C|} \tag{7.13} $$

Equation (7.14), shows that maintaining the average of the individual Q-values is equivalent to maintaining the values themselves. However, this approach may not be the most fair, in that large rewards for an individual player could dominate the smaller (possibly negative) rewards for other coalition members.

$$ {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P \in C} {}^{\tilde{\mu}}_{P}Q^{\mu}(s,o,\tilde{\pi})}{|C|} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P \in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{|C|} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P \in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} \right] + \sum_{P \in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{|C|} \tag{7.14} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = \frac{\sum_{P \in C}\left[{}^{\tilde{\pi}}_{P}r^{o}_{s} \right]}{|C|} + \frac{\sum_{P \in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{|C|} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \frac{\sum_{P \in C}\left[\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{P}v_{s'} \right] \right]}{|C|} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \frac{\sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')\sum_{P \in C}\left[{}_{P}v_{s'} \right] \right]}{|C|} $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')\frac{\sum_{P\in C}\left[{}_{P}v_{s'} \right]}{|C|} \right] $$

$$ {}^{\tilde{\mu}}_{C}Q^{\mu}(s,o,\tilde{\pi}) = {}^{\tilde{\pi}}_{C}r^{o}_{s} + \sum_{s'}\left[{}^{\tilde{\pi}}f^{o'}(s,s')_{C}v_{s'} \right] $$

7.2.5 Fairness and Reward Allocation

In general, when it comes to determining the allocation factor φ for a given option, what we would like it to represent is the amount of the option that is allocated to an individual participant within a coalition. Therefore, in a perfect world:

$$ {}^{\tilde{\pi}}\phi_{s}(o,\Psi) = \frac{{}^{\tilde{\pi}}_{\Psi}r^{o}_{s}}{{}^{\tilde{\pi}}_{C}r^{o}_{s}} \tag{7.15} $$

This mechanization incorporates information about the actions of the non-coalition players (represented by $\tilde{\pi}$) as well as the current state of the agent. This means that determining the allocation of coalition utility to agents is not as straightforward as described here (i.e. “Player 1 gets 50% of the reward”), however, for this assessment we have implemented the naïve approach.

To address the limitations of using a fixed approach to allocating future expected value, in future work, there would be benefit to maintaining some knowledge of the individual utility (at least for your own utility) to assess whether the utility being allocated to an individual aligns with the utility they independently calculate that they deserve (or would receive if ϕ allocated perfectly). If these utilities are out of alignment, it is unclear that an agent would pursue this option independently.

To encourage another agent to participate in an option for which the allocation may be less than fair, a reward redistribution mechanism could also be implemented, allowing the agents who would receive greater than their fair share to incentivize the other agents to participate. Additionally, updates to the initiation set could be made to support agents disregarding options where either properties of the current state or the actions of the non-coalition agents cause the option itself to be unfair.

7.3 Experimental Structure

The goal of the combination of rewards is to be able to use the values maintained for coalitions in a reduced format, while assuring that any equilibrium found in the reduced format game would also be an equilibrium in the fully populated game. To experimentally assess that, we leveraged the grid world described in Section 5.1, with a few additions. Because we looked at three players rather than two, we added the third player in with a starting position of [0,2] and a player goal position of [0,2]. In order to ensure that the system was fair to all three agents, the third player's goal for their flag was in square [0,1]. This ensured the third player could not immediately drop the flag after moving down from the top row, but would have to drop the flag and then move to their goal state. The three players remain in competition for two flags.

This world is similar to the one described in Chapter 5, and is illustrated in Figure 7.3. Agents are responsible for claiming a flag (which they can only do if they (a) share the square with another agent and the flag and (b) do not attempt to simultaneously retrieve the flag with the other agent) and bringing it back to the player-specific goal location. Once a flag has been dropped in the player specific goal location and the player has reached their goal location, the player gets 100,000 points. Each player also gets a bonus for possessing a flag. All players get a smaller bonus for each flag possessed when any player possesses a flag. Player 2 has a separate goal location and flag location to ensure that all agents start with the same opportunity to reach the goal state.

These experiments used Nash SARSA (described in section 2.1.5) as the learning technique to update Q-values. Experiments were run for 10,000 episodes. Execution of an episode continued until a player reached a winning state or 100 timesteps were executed. In the learning period, players would randomly select an option per an exponentially decaying rate (λ) of -0.0005, which would transition to a 1% exploration rate (ϵ) after the learning period was completed (episode 8400). Initial Q-values were set to 0.

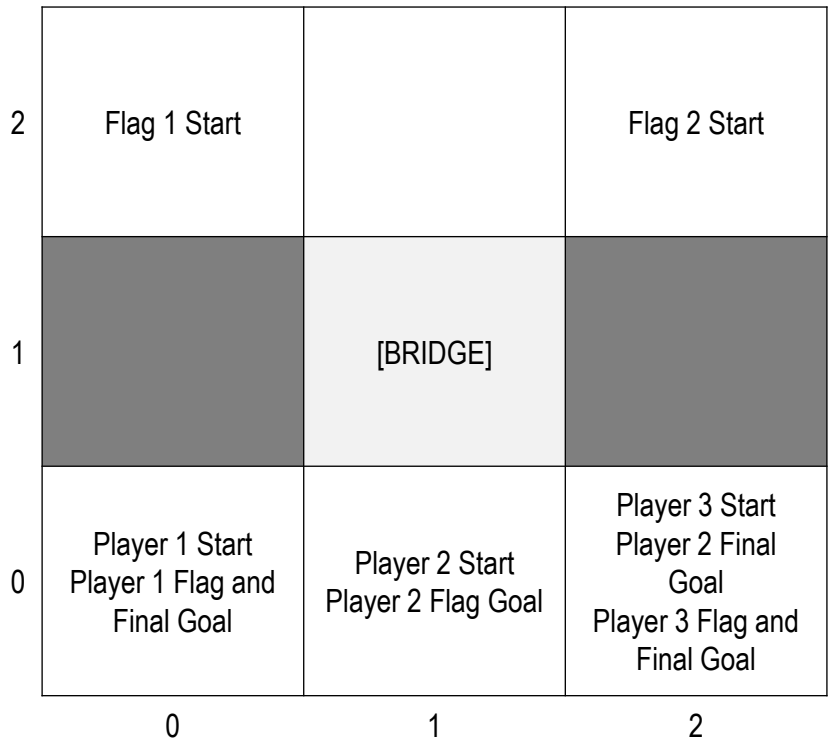

Figure 7.3 The Capture the Flag Grid World

This grid world supports a game where three players compete to retrieve two flags from the upper row, and return them to goals in the lower row. Although the game ends as soon as the first player reaches their goal, players do have to collaborate to cross the bridge and to retrieve the flags.

In the initial formulation, there were five options available to players in this world. Four of them were similar to those described in Chapter 5 (move up, move down, get Flag 1, get Flag 2), and the final option was a default option, existing solely in states where there are no other options available to provide a means to partition the world into two pieces.

7.3.1 Importance of Multiagent Option Availability

The example test case described in this section highlights the importance of providing a varied library of multiagent options to the agents, and ensuring that the library covers the entire state space. Some limitations of the initial game definition described above are shown in Figure 7.4. Because of the structure of the reduced game and the definition of the initiation steps of the available multiagent options, agents can get into scenarios where there are no “good” multiagent

options to select, but due to the partitioning requirement, one must be selected anyway. Image (a) depicts a state where two agents have retrieved flags. At this point, the highest-value next objective would be to move down to the lower part of the grid world to deposit their flags in the goal locations. Luckily, in this state, their only available multiagent choice is to team with another agent and execute the move down multiagent option. Because of the construction of the game itself, at least two of these agents must team up to execute this action. Image (b) shows a state that would be the result of the white and gray agents executing that option. Both agents are now on the bridge square, and seeking to move down to arrive at the goal locations. However, the only multiagent option available to them in this state is the default option, which has a default policy of consistently moving up. At least one of the two agents on the bridge square must select this option, and therefore move away from their goal (and the highest value reward). To compound the suffering, the only multiagent option available on the bridge square is the default option, so achieving the desired end state requires the other agents (i.e. black and gray) to join together repeatedly, so that the agent in a coalition of 1 (white) has the ability to select the desired primitive actions until reaching the goal. This illustrates the reality that in order to utilize this approach to reduce the

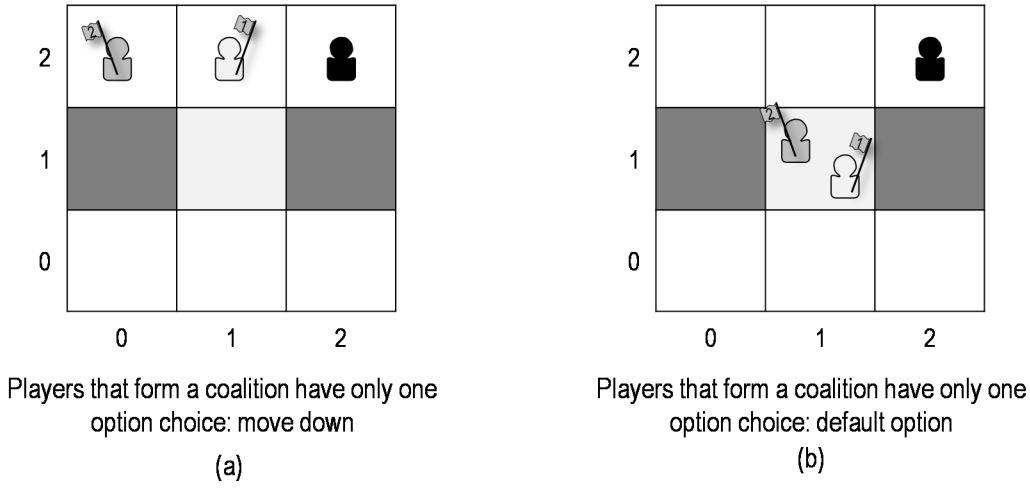

Figure 7.4 Limiting Scenarios in the Capture the Flag Grid World

If a rich and varied selection of multiagent option choices are not available to the agents, the policies they learn could be suboptimal.

complexity of the associated games, there must be a richly populated set of multiagent options from which agents can select behaviors available in every state.

This issue was addressed by increasing the available options for coalitions, and thereby providing a deeper library of behaviors available to teams of more than one agent. To accomplish this, the move up and move down options were extended to move the agents beyond the bridge (instead of the previous approach, which terminated when agents reached the bridge) and three new collaborative "go home" options were included in the available option set for the lower row of the world, one for each pair of home locations. The resulting option definitions for the capture the flag world are shown in Table 7.1. The total number of MDP states in the Capture the Flag World is 944,784. Once these collaborative options were added to the system, stage game state sizes for the reduced format stage game ranged from 140 to 440 game states.

Table 7.1 Option Initiation States and Termination Probabilities for the Capture The Flag World

This table illustrates the initiation states associated with the eight two-player multiagent options, as well as the definition of termination probabilities.

Option	*Initiation State Definition*	*Termination Probability Definition*
Move Up	$\psi_{A,y} = 0 \wedge \psi_{B,y} = 0$	$\begin{cases} 1 & \psi_{A,y} = 2 \wedge \psi_{B,y} = 2 \\ 0 & otherwise \end{cases}$
Move Down	$\psi_{A,y} = 2 \wedge \psi_{B,y} = 2$	$\begin{cases} 1 & \psi_{A,y} = 0 \wedge \psi_{B,y} = 0 \\ 0 & otherwise \end{cases}$
Role A Gets Flag 1	$\psi_{A,y} = 2 \wedge \psi_{B,y} = 2 \wedge \psi_{A,p} = NONE$	$\begin{cases} 1 & \psi_{A,p} = Flag1 \\ 0 & otherwise \end{cases}$
Role A Gets Flag 2	$\psi_{A,y} = 2 \wedge \psi_{B,y} = 2 \wedge \psi_{A,p} = NONE$	$\begin{cases} 1 & \psi_{A,p} = Flag2 \\ 0 & otherwise \end{cases}$
Role A Goes To Home 1 Role B Goes To Home 2	$\psi_{A,y} = 0 \wedge \psi_{B,y} = 0$	$\begin{cases} 1 & \psi_{A,y} = Home1 \vee \psi_{B,y} = Home2 \\ 0 & otherwise \end{cases}$
Role A Goes To Home 1 Role B Goes To Home 3	$\psi_{A,y} = 0 \wedge \psi_{B,y} = 0$	$\begin{cases} 1 & \psi_{A,y} = Home1 \vee \psi_{B,y} = Home3 \\ 0 & otherwise \end{cases}$
Role A Goes To Home 2 Role B Goes To Home 3	$\psi_{A,y} = 0 \wedge \psi_{B,y} = 0$	$\begin{cases} 1 & \psi_{A,y} = Home2 \vee \psi_{B,y} = Home3 \\ 0 & otherwise \end{cases}$
Default Option	When none of the above are true	$\begin{cases} 1 & \psi_{A,y} = 0 \vee \psi_{A,y} = 1 \vee \psi_{A,y} = 2 \\ 0 & otherwise \end{cases}$

For this experiment, we ran 100 time steps with an exploration factor of $\varepsilon = -0.0005$. At the conclusion of 10,000 episodes, we performed analysis on the current state of the world by evaluating all visited game states and generating both the reduced form and the fully populated game based on the learned Q-values. We then computed the equilibrium for the reduced form game and analyzed those equilibria to see if they were also equilibria in the fully populated game. The assessments presented in the following sections evaluate the solutions on the basis of two main criteria: convergence to a given learned policy, and the correspondence of equilibria found in the reduced format table with the extended form table.

7.3.2 Approaches to Learning

In these experiments, we will describe two alternatives to selecting actions. The first alternative, referred to hereafter as the "individual" case, maintains utilities for players individually, and then combines those utilities into the reduced format game to select an action. When we assess equilibria correspondence between the reduced format games and the fully populated game, we will use the player-based Q-tables to build the reduced format game. In the other alternative, we combine and update player rewards into coalition rewards as we are learning. This use case is referred to as the "incremental" use case and the process to evaluate these combination methods is detailed in Section 7.5.

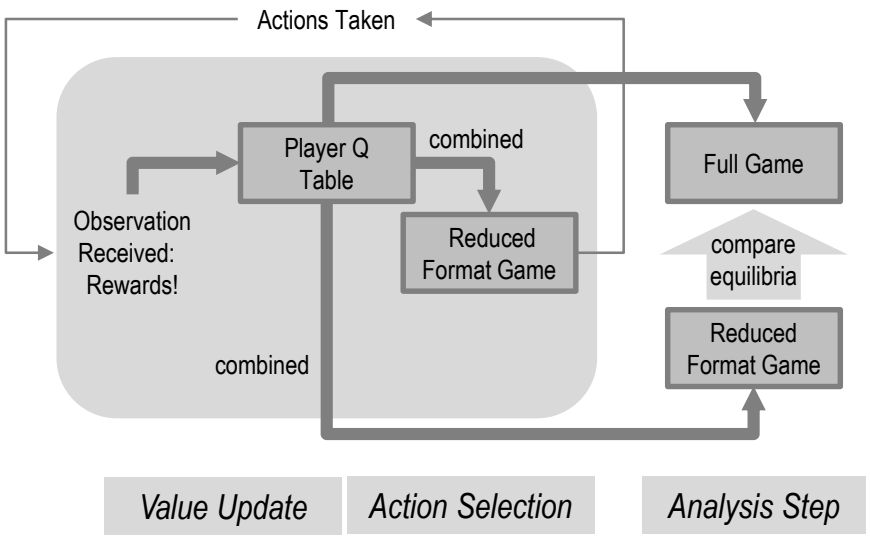

Figure 7.5 Learning and Analysis Process for the Individual Use Case

The assessment termed the "individual" use case is one where individual player utility tables are maintained. Values from those tables are combined to generate the payoffs for the reduced format game, which is where actions are selected. After the learning phase is complete, the equilibria generated from the reduced format games (again, constructed from the individual tables) are compared against the full game generated from the values in those same tables.

7.4 Individual Case

In these experiments, while learning, we maintained Q-table values for each player (as is standard) and updated those tables per the traditional Q-value Nash SARSA update. We then evaluated two different mechanisms to populate the reduced format game. We combined the values from the player-specific Q-tables to populate the reduced format table in order to select actions. After the experiments completed, we evaluated the equilibrium generated from the reduced format table against the fully populated (three-player) table to assess if equilibria in the smaller table were also equilibria in the larger table.

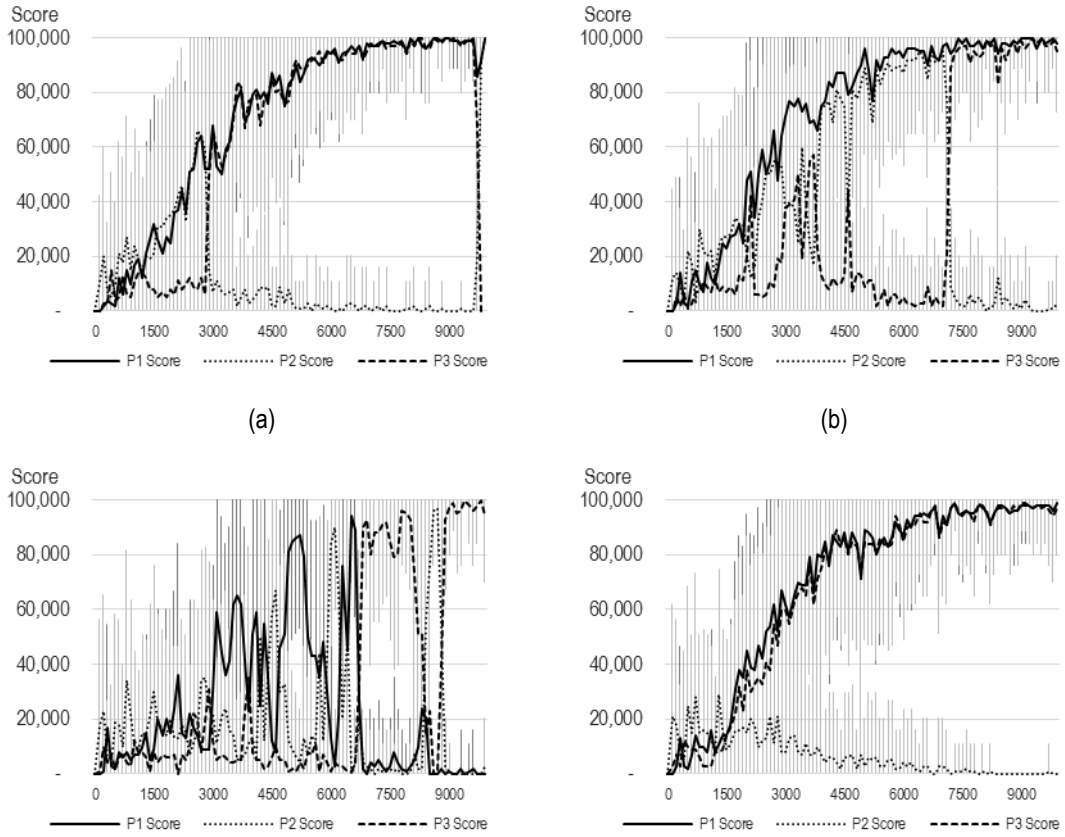

Figure 7.6 Learning Results in the Individual SUM Use Case

While agents do converge to policies, the symmetric nature of the problem results in some switching of equilibria during the experiments (shown as the full height vertical lines). In addition, case (c) converged to a single-player only solution.

7.4.1 Learning Results

This problem is symmetric, and so the results did show some vacillation of choice in terms of which coalition to form. This makes it difficult to represent these results as a "hi-medium-low" player graphic. For example, the results from four individual runs for the sum case are shown in Figure 7.6. As you can see, three of the four runs converged to a two-player solution, but two of those three switched coalitions late in the process.

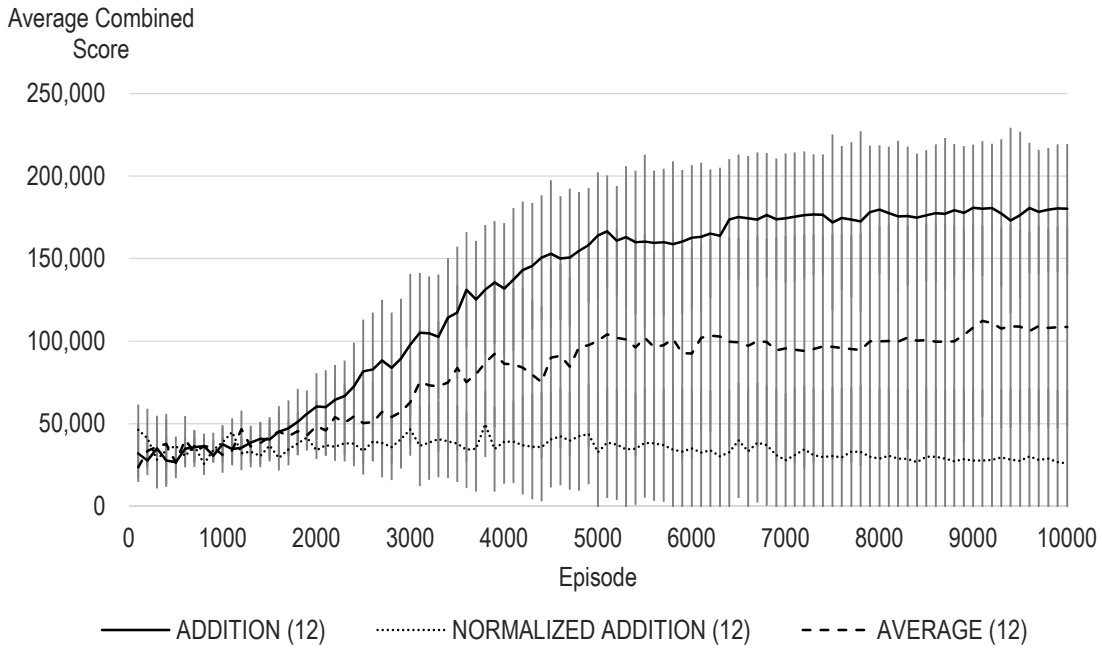

Figure 7.7 Average Combined Score for the Individual Use Case

In the individual game, both the average and addition cases converged to solutions where at least one player would retrieve and return home with the flag. The normalized addition case did not, but may have been penalized due to the resolution of the equilibrium solver. The number of experiments run for each case (12) is displayed next to the legend label.

Due to the switching of teammates discussed above, the charts displayed in Figure 7.7, comparing the results of each of the combination mechanisms studied aggregate the scores of all three players together. It is worth repeating that in this use case, these utilities are independently maintained and combination of utility only occurs to generate the reduced format table (when selecting the next option to execute). The addition case consistently converged to an optimal solution, where both players would simultaneously return their flags to the goal state. The average case was able to discover a solution where a single player was able to retrieve their flag and deposit it, but was not consistently able to discover the more optimal joint solution. The normalized addition case was not consistently able to reach either winning case. On further analysis, it was discovered that 17% of Q-values in the normalized addition case were below the fidelity of the GAMBIT solver (set to 6

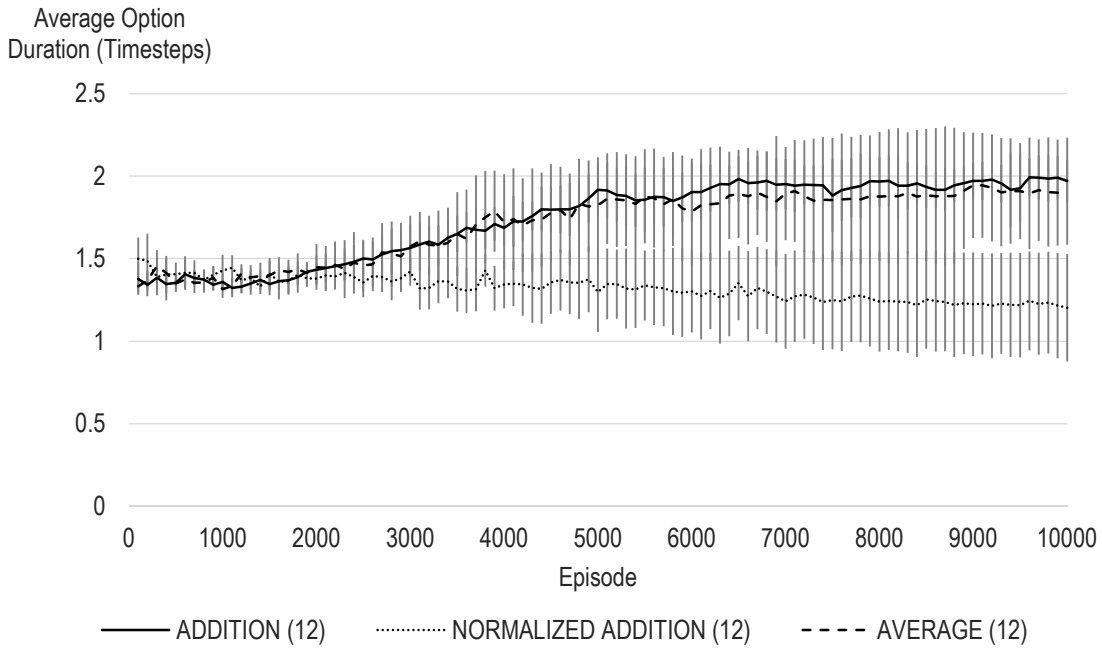

Figure 7.8 Average Option Duration for the Individual Use Cases

In general, the average and addition cases both execute options for at least one step after the initial selection of the option. The normalized addition case executes options for a shorter number of timesteps.

decimal places). It is possible that some equilibria were miscalculated or invalidated due to these very low payoffs.

Also of interest is the number of timesteps options executed before terminating. The duration of options depends heavily on the termination criteria used to assess when an option has been completed. For these experiments, since the selection of options was centralized, the criteria used was to terminate based on the criteria established in the option itself. Selection of primitive options were not included in the calculation. Figure 7.8 shows the three cases. Both the average and addition cases typically execute options for at least one step after their initial action. Because the termination criteria for the default option includes every state, that option always terminates after a single step. The other (non-default) options terminate per more restricted criteria, and typically execute for more than one step, so a larger number of steps executed here indicates that the agents

are taking advantage of the full set of behaviors available to them. The normalized addition case, however, does not exhibit that same result.

7.4.2 Assessment of Equilibria

After the completion of each run of these experiments, for each state that was updated at least once by the learning algorithm, both the reduced format game and the fully populated game were constructed out of those values. Equilibria were calculated for the reduced format game and assessed to see if they were also equilibria in the fully populated game. The results of this assessment for the different combination approaches are shown in Figure 7.9. The addition case correlated almost perfectly between the two sets of tables. In that case, there were only four states out of 944,784 possible states where there were equilibria in the reduced format table which were not equilibria in the fully populated table, and the utility discrepancies in those cases were all very small. While many more cases did not match in the normalized sum case, the average deviation utility for those cases was 0.0009, again pointing to potential numerical issues with the small utility values. The average case, while showing some disparity, also indicated a deviation delta of negative values, which would indicate some sort of computation error in that case.

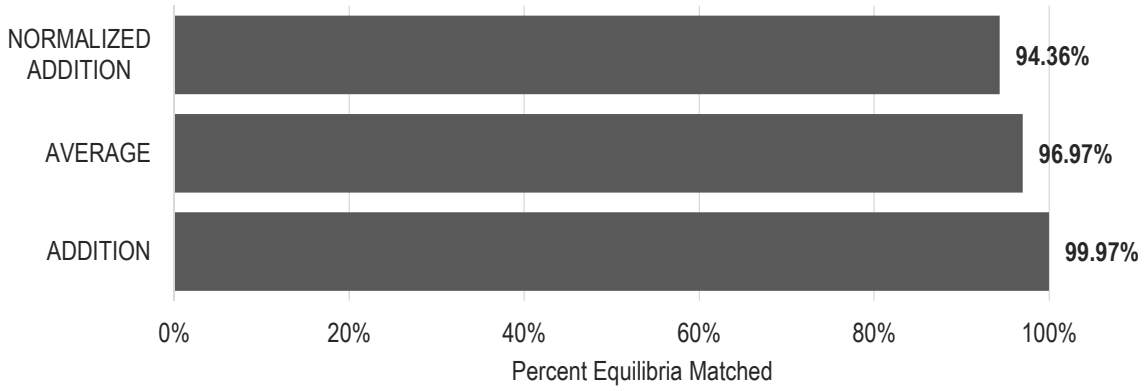

Figure 7.9 Equilibrium Correspondence between the Reduced Format Game and the Game Generated From Individual Values

In comparison of the equilibria values found in the reduced format games as opposed to the more traditionally structured games, the addition case generated a significant number of matches, and the other cases, while not as well aligned, were also very close.

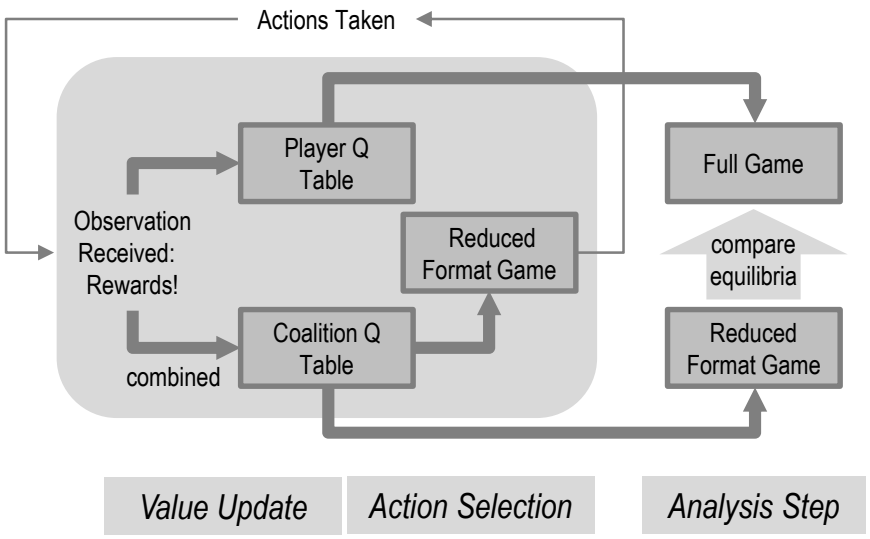

Figure 7.10 Learning and Analysis Process for the Incremental Use Case

The assessment termed the "incremental" use case is one where both individual player utility tables and coalition utility tables are maintained. Values from the coalition tables are updated based on coalition rewards and used to provide the payoffs for the reduced format game, which is where actions are selected. In addition, player Q-table values are also updated (but are not used for learning). After the learning phase is complete, the equilibria generated from the reduced format games which are constructed from the coalition Q-tables are compared against the full game generated from the values in the player Q-tables.

7.5 Incremental Case

In the incremental experiments, while learning, we maintained two sets of Q-table values. The first set of Q-tables (one for each player) was updated per the traditional Q-value Nash SARSA update. The second set (one for each coalition) was updated with values calculated per one of the mechanisms described above (sum, average, etc). As the agents learned, we updated the coalition-specific Q-tables with combined values. We then used those values directly to populate the reduced format table in order to select actions. When these experiments completed, we evaluated the equilibrium generated from the reduced format table against the fully populated (three-player) table to assess if equilibria in the smaller table were also equilibria in the larger table.

7.5.1 Learning Results

Since this is still a symmetric scenario, the charts displayed in Figure 7.11, comparing the results of each of the combination mechanisms studied aggregate the scores of all three players together. It is worth repeating that in this particular set of experiments, the utilities of coalitions are maintained and used to select the next option to execute from the reduced format table. Utilities of individual agents are also maintained for the purpose of comparing equilibria across the reduced format and the fully populated game. The addition case did the best in terms of overall outcomes in this approach, returning the optimal two player solution in the majority of runs executed. The average case performed slightly less well, but still frequently returned the optimal two-player-win solution. The normalized addition case fared the worst, but still was able to frequently converge to a single player winning solution.

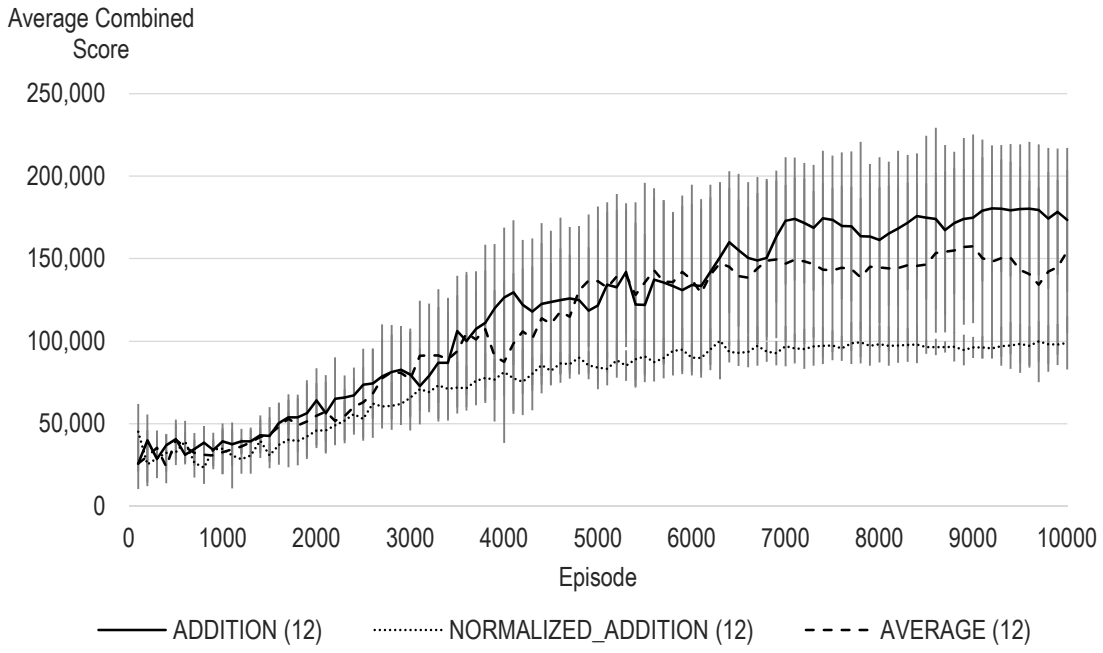

Figure 7.11 Combined Player Score for the Incremental Use Case

The addition case fared the best in terms of collaborative player outcome, followed by the average case and finally followed by the normalized addition case.

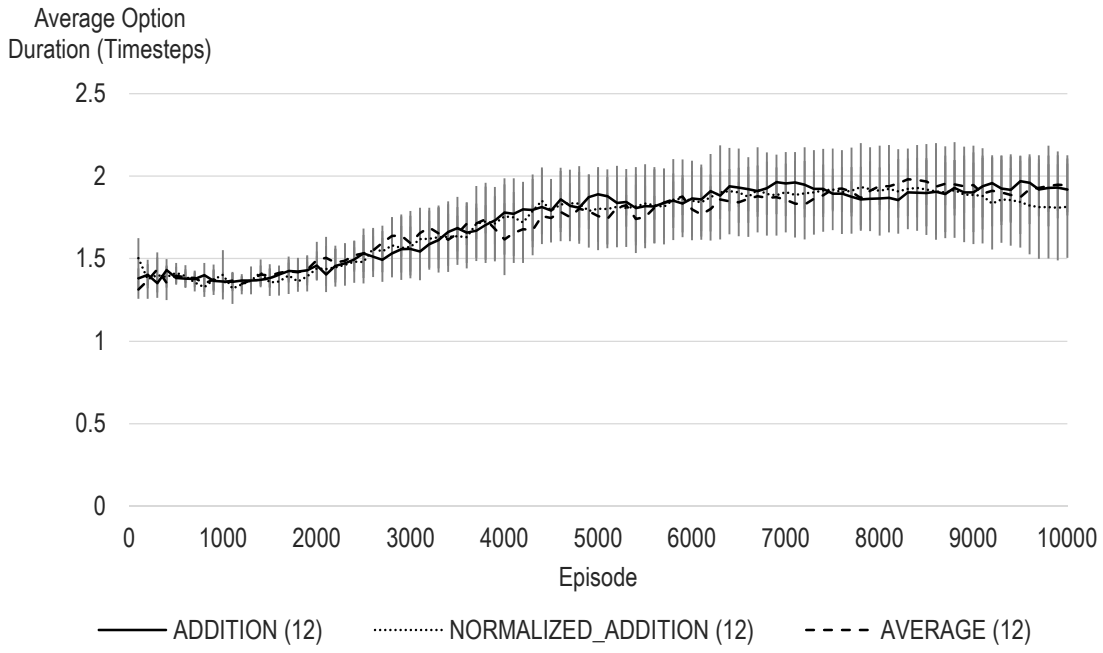

Figure 7.12 Average Option Duration for the Individual Use Cases

In these cases all of the three methods utilized the non-default options in a similar way, executing options on average about .9 timesteps after the initial timestep.

The duration of options depends heavily on the termination criteria used to assess when an option has been completed. For these experiments, since the selection of options was centralized, the criteria used was to terminate as the policy associated with the option deemed appropriate. Selection of primitive options were not included in the calculation. Figure 7.12 shows the three cases in terms of their average option duration over time. When the players updated coalition utilities independently, all of the players executed options for nearly one step after the option was initiated.

7.5.2 Assessment of Equilibria

After the completion of each run of these experiments, for each state that was updated at least once by the learning algorithm, both the reduced format game and the fully populated game were constructed out of those values. Equilibria were calculated for the reduced format game and assessed to see if they were also equilibria in the fully populated game. The results of this assessment for the different combination approaches are shown in Figure 7.13. All three of the methods perform very close to perfectly in this regard, with less than 1% of the equilibria calculated deviating from the full game. While these equilibria deviate to a greater degree than those populated directly from the individual tables, they are also calculated using the heuristically generated allocation factor ϕ. This means that in cases where the fixed allocation factor is unfair (for example, helping another agent get to their base before you collect your own flag), the utility planned may over-estimate the actual reward received by agents. Implementing either (a) a check to assess if options are fair as part of an expanded initiation set criterion or (b) a mechanism to share rewards among agents to make multiagent options more equitable should alleviate some of this discrepancy. In addition, having each agent maintain their own utility values and use those in

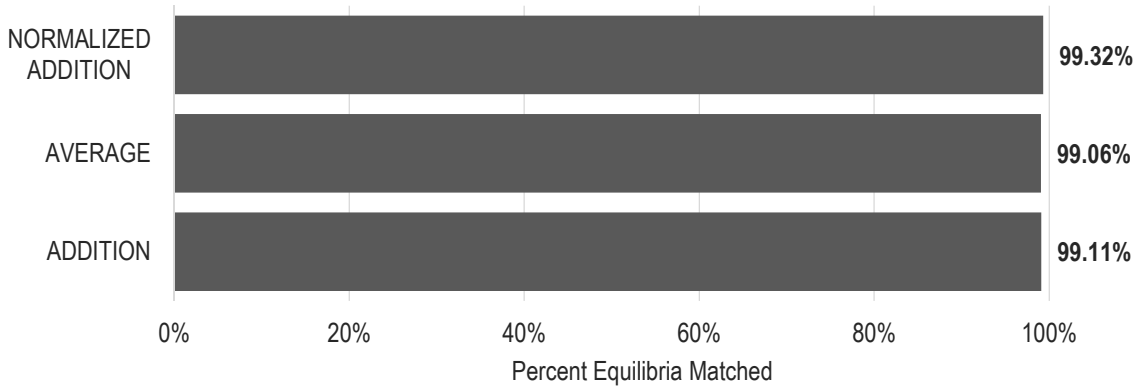

Figure 7.13 Equilibrium Correspondence between the Reduced Format Game and the Game Generated From Incremental Values

In comparison of the equilibria values found in the reduced format games as opposed to the more traditionally structured games, all cases performed very close to perfect in terms of equilibria matching.

Table 7.2 Summary of Experimental Results

The experiments run for the individual and incremental test cases are summarized below.

Case		***Convergence***	***Equilibria***
Values Constructed From Individual Tables	**Sum**	Converged to the optimal solution.	**99%** -- Significant Equilibria Match. Discrepancies likely due to numerical calculation
	Average	Converged to the optimal solution in some cases, a single player solution in others.	**96%** -- Good Equilibria Match. Discrepancies likely due to numerical calculation
	Normalized Sum	Did not converge to the optimal solution. Some indications of numerical discrepancies.	**94%** -- Good Equilibria Match. Discrepancies likely due to numerical calculation
Values Maintained Incrementally	**Sum**	Converged to the optimal two-player solution.	**99%** -- Significant Equilibria Match. Some discrepancies potentially due to allocation factor.
	Average	Converged to the optimal two-player solution.	**99%** -- Significant Equilibria Match. Some discrepancies potentially due to allocation factor.
	Normalized Sum	Converged to a single player solution in most cases.	**99%** -- Significant Equilibria Match. Some discrepancies potentially due to allocation factor.

conjunction with the coalition rewards to automatically generate the allocation factor could also benefit these results. These extensions to the work described in this dissertation are further discussed in the future work section.

7.6 Summary

A summary of the results discussed in this chapter is presented in Table 7.2. Overall, the maintenance of coalition values instead of individual values shows promise, however, the heuristic approach to allocate those values can result in an allocation of value that over or underestimates the actual reward of the agent. Mechanisms to allocate this value more appropriately to individual agents will be valuable extensions to this work.

CHAPTER 8

Decentralized Coalition Selection

In previous chapters, the mechanisms discussed have been evaluated on choices made by a centralized agent. While this simplifies the analysis of equilibrium commonality, to truly apply this mechanism in more realistic scenarios we must assess the outcomes when we allow agents to select actions independently.

8.1 Impact of Independent Action Selection

Under this construct, when agents select actions independently, it is possible that they will each individually select a valid partition that results in the aggregate in an invalid partition of the world. As a simple example, consider if all agents select to be individual actors, expecting the remaining agents to be allied in a coalition. Should this happen, the aggregate action set returned will include primitive bound option choices for all the agents, which will be more choices than can be represented in the table. Because of this, a determination must be made as to how the results of that choice are updated through the learning process within the constraints of the reduced format game. Mechanisms considered included: performing no update for invalid choices, updating all bound options that included those primitive selections as primitive steps (similar to the state assessment termination criteria), or updating an additional option choice for actions not in the table. If no update is performed when an invalid partition is jointly selected, the outcome of learning should be similar to the outcomes described in the previous chapter, differing only in the rate of convergence (disregarding data should result in more samples required to achieve policy convergence). On the other hand, as reflected in the previous work on state assessment, updating all aligned partitions based on the actual behavior could increase the rate of convergence, but requires significant knowledge about the inner workings of coalitions to be achievable. Providing another bound option, however, to reflect the outcomes when unaligned choices were made, holds promise. This approach is more fully discussed in the next section.

8.2 Placeholder Bound Options

In order to handle the case where all of the selected actions are not present in the reduced format table, an additional "placeholder" option was added. This option is an apathetic option, because it reflects a reality where the decision making agent either does have enough information or has no preference to the behavior of the other coalitions. This placeholder option was added once for every "column" coalition into the reduced format table, as shown in Figure 8.1. If the aggregate selected actions did not align with a valid partition of the world, the agent's selected action/coalition would be updated with the other partition of the world being represented by the placeholder option. For example, if every agent chose to act independently while the other two agents formed a coalition, the selected aggregate coalition set would be $\{P_1, P_2, P_3\}$. In the previously formulated structure, that particular coalition set would not exist in the game, and the value would be φ. The formulation specific to distributed option selection, however, allows each player (in this case P_1) to update their selected action against the placeholder as the following coalition set $\{P_1, P_2 P_3\}$. Note that because

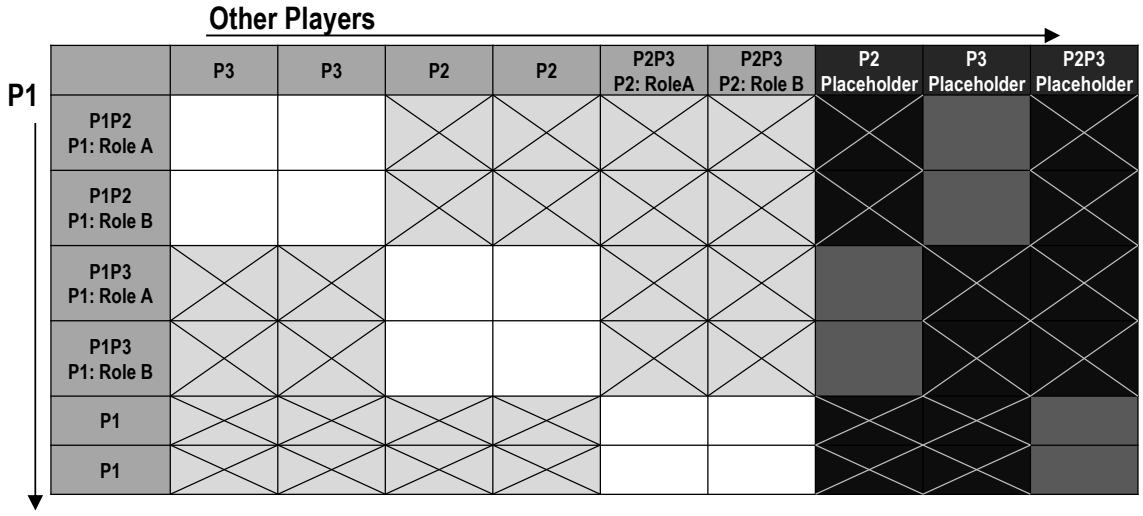

Figure 8.1 The Structure of the Matrix Game with "Placeholder" Options Added

When agents select actions independently, it is possible that they will return an aggregate choice that is not a valid partition of the game. In order to effectively utilize this information (as opposed to disregarding it), "placeholder" options were created. These options, which have no associated policies or roles and therefore can only be selected as an option other agents play, serve as aggregators for decisions made that are unable to be represented in the reduced format table.

these actions are added to the columns (when the number of partitions $\mathrm{H} = 2$) of the reduced format table, they act as aggregators for any other agent's possible actions and represent the current strategy of the other agents. When an individual agent selects an action, that agent will never select the placeholder option for their own execution, but they could select it as a valid strategy for some or all of the other players. The addition of these options may serve to ameliorate some of the issues with partitioning in general – namely, this structure allows every agent to select options against a composite best response of the other agents.

8.2.1 Formulation of the Stage Game

Each agent p independently selects an action based on the updated game shown in Figure 8.1. This game is similar to the one defined in Section 6.1.6. We define a placeholder bound option ${}_{c}B_{s}^{*} = \{c, o^{*}, X^{*}\}$ where for option $o^{*} = \{\Psi, \pi, \tau, I\}$, $\Psi^{*} = \{\varnothing\}$, $\pi^{*} = \{\varnothing\}$, $\tau = 1 | s \in S^{+}$ and $I = S$. Note that placeholder bound options have no associated behavior or role assignment, and exist solely to provide a mechanism to link utility to a coalition. Stage games associated with each state of our stochastic game are now defined as $G^{*} = \{\mathrm{H}, B_{s}^{+}, u\}$, where H is the number of allowed partitions of the world (again, in following discussions and examples, for simplicity, we assume $\mathrm{H} = 2$) and $u = (u_{1}, \ldots, u_{\mathrm{H}})$ is the set of utilities (or payoffs) for each coalition set. $B_{s,1}$ is defined as the first set ("row player") of bound options available to initiate in state s where the agent selecting the action (p_{1}) is a participant ($p_{1} \in c_{B}$). For all other coalitions n within the coalition set, $B_{s,n}^{+} = \{B_{s,n}, B_{s,n}^{*}\}$. The total set of bound options B_{s}^{+} used in the stage game is defined as in Equation (8.1).

$$B_{s}^{+} = \{B_{s,1}, B_{s,2}^{+}, \ldots, B_{s,\mathrm{H}}^{+}\} \tag{8.1}$$

Actions are selected from the reduced format game based on an equilibrium calculation, with no changes from the previously discussed approach (other than the addition of placeholder bound actions creating a larger game space).

Table 8.1 Learning Updates with Placeholder Options

When players do not select in aggregate to play an option which can be represented in the reduced format table, each player will update their action against the corresponding placeholder option.

Player	*Player Selected Bound Option Set*	*Player Updated Bound Option Set*
Player 1	P1: up P2P3: default option [Role A: P2 Role B P3]	P1: up P2P3: placeholder []
Player 2	P1P2: Get Flag 1 [Role A: P1 Role B P2] P3: up	P1P2: Get Flag 1 [Role A: P1 Role B P2] P3: placeholder []
Player 3	P3: up P1P2: default option [Role A: P2 Role B P1]	P3: up P1P2: placeholder []

8.2.2 Learning Updates

When agents receive observations from the environment, those observations include all of the selected actions from all other agents. Those actions are then combined to create the as-executed partitioning. An example of this is presented in Table 8.1. In this example, the combination of agent actions selected (shown in the second column) is not a valid partition of the environment. Since this is the case, the value that will be updated by each agent (shown in the third column) will include the placeholder option as the corresponding disjoint bound option selection.

The learning update itself is the same as that described in Equation (6.11), only differing in the inclusion of the additional placeholder option choices.

8.2.3 Allocation Factor Refinement

When considering a distributed scenario, we also must re-evaluate mechanisms to allocate future coalition value to coalitions. The major issue with the allocation factor as previously defined is that if the option does not terminate as planned, the agents involved could propagate significantly different values than they receive. For example, as shown in Figure 8.2, once Option 1 completes, the value that will be used as the expected value of future states will come from Option 2 and the white agent's future states. If ϕ_B is 50%, but the black agent really only receives 10% of the rewards associated with that option, then the value of Option 1 will also include a contribution from the gray player – who did not participate in Option 1. In addition, the value of the gray agent's policy will be similarly diminished. In order to address this, all individual player's utilities were also maintained with the combined utility. While the combined utility was used to calculate the payoffs, when

allocating rewards from future coalitions, the individual utility was used instead of the allocation factor.

8.3 Experimental Results

8.3.1 Experimental Scenario

The scenario used to evaluate the placeholder options is the same scenario described in detail in Section 7.3.

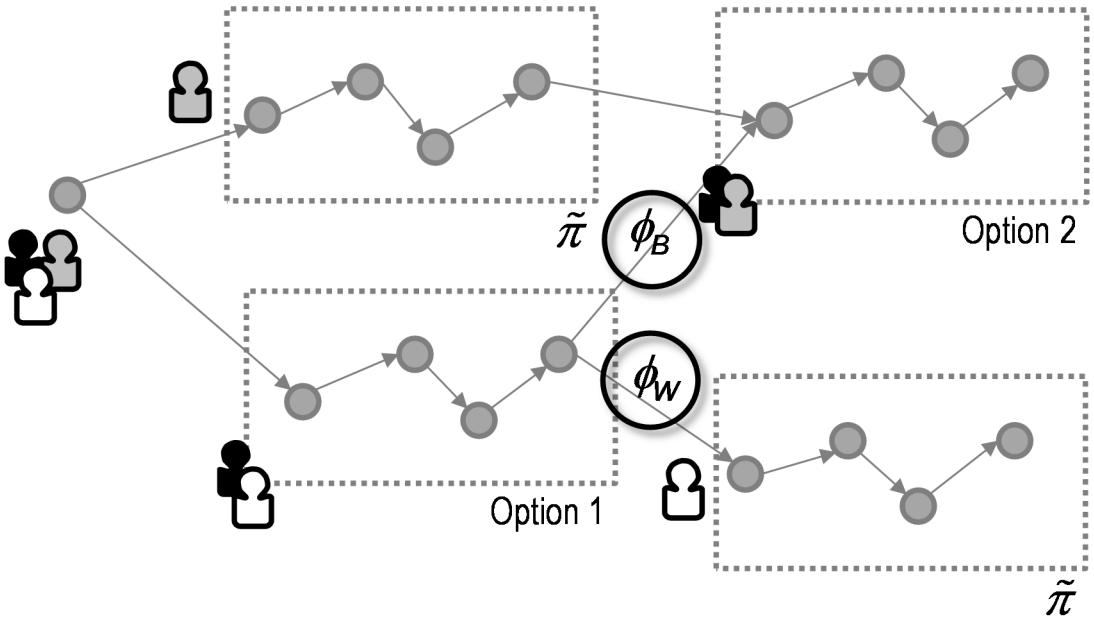

Figure 8.2 Allocation Factors

When considering the propagation of utility from future states, if the allocation factor ϕ does not represent the true allocation of rewards, behavior can be unexpected.

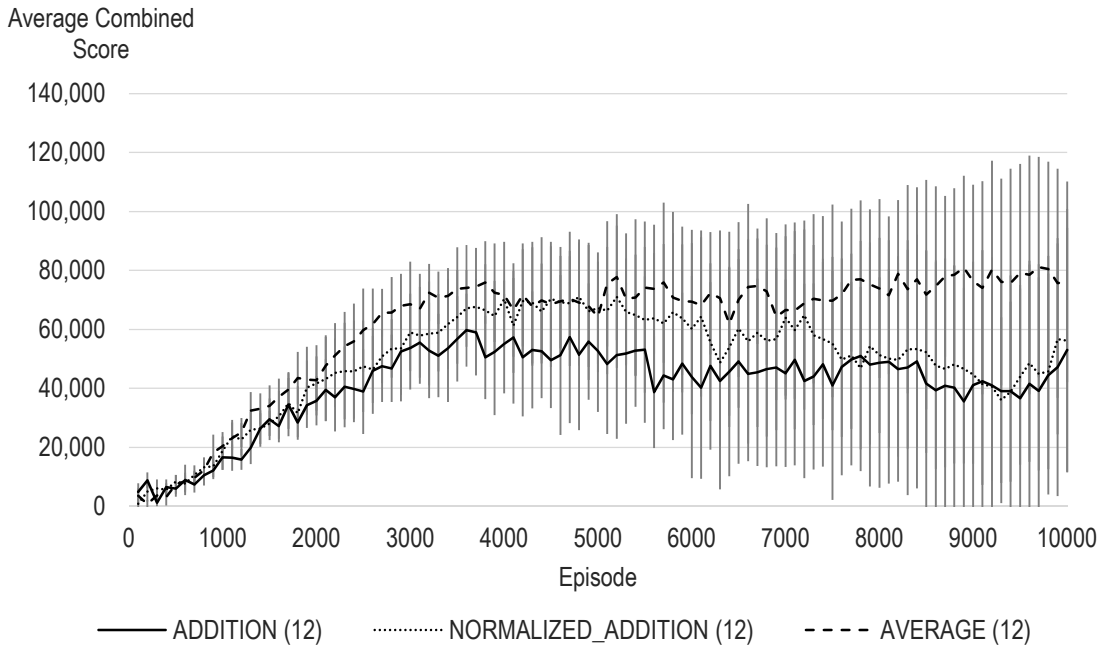

Figure 8.3 Convergence of the Distributed Algorithm

When the option selection is distributed across agents, the policies do not converge to a jointly optimal solution.

8.3.2 Learning Results

As shown in Figure 8.3, when applying this method in a distributed fashion, the policies generated by the agents independently converge to suboptimal individual solutions, however, they do not converge to a joint solution where both players receive and return flags. These convergence issues are not unusual when employing these types learning algorithms (NashSARSA) in distributed environments, as the coordinated selection of consistent equilibria is critical to enabling agents to find optimal joint solutions.

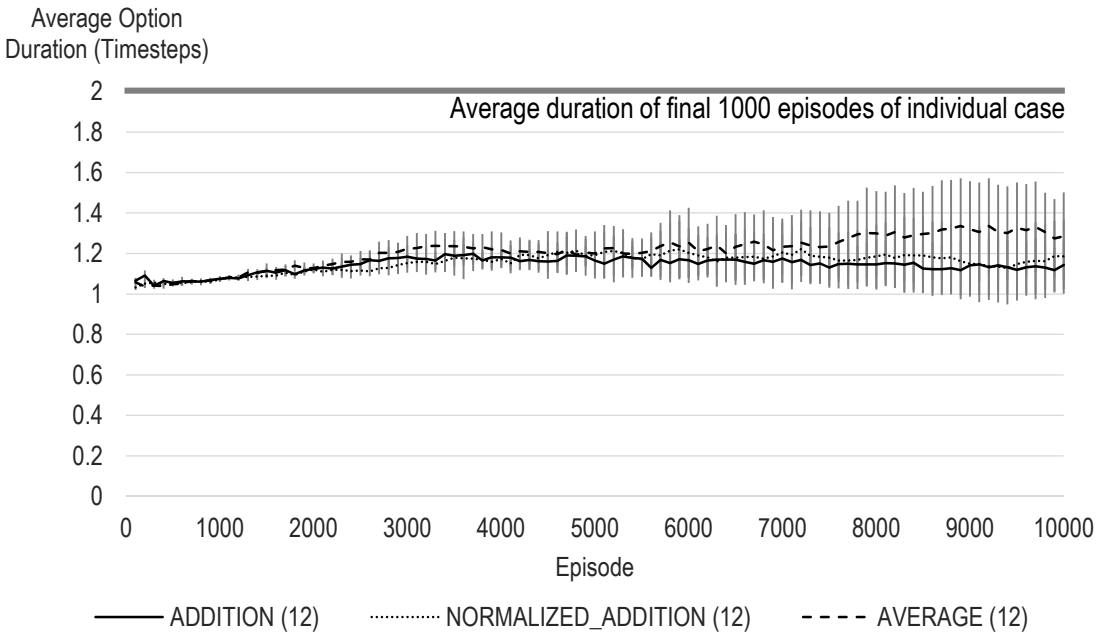

Figure 8.4 Average Option Duration

The average number of timesteps agents executed a multiagent option in the distributed case was significantly less than in the individual or incremental cases.

Part of this may be due to the termination criteria associated with executing these options. Figure 8.4 illustrates that the average time players execute an option in the distributed approach is less than one time step. In the individual and incremental centralized cases, players executed options for nearly one timestep on average. The termination criteria for these runs was set to the explicit approach, where players would only continue to execute an option if the other participants in their coalition were also executing the same option. When selecting options in a distributed approach, the players would (especially early on when exploration was high) frequently not select the same bound option to execute. This would have two results in the new structure: (1) it would terminate the option after the first action is taken (here shown as 0 timesteps) and (2) it would update the selected option against the placeholder option. The consequence to immediate termination is that the option itself is not enabling the players to reach distant rewards.

8.3.3 Termination Criteria

In order to assess the impact of different termination criteria on the distributed case, four different approaches from the set discussed in Section 5.5 were implemented: terminate if all participating players did not explicitly select the same bound option (explicit), terminate if all participating players are not executing actions in line with the same bound option (state assessment), terminate when the policy ends or after 5 timesteps, and terminate when the policy ends or with 20% probability. The "explicit" case is the case that has been discussed up until this point in the dissertation (as termination analysis at all makes less sense for a centralized pick). Figure 8.5 illustrates the differences between the learning outcomes for the different termination criteria (all combining values using the addition method).

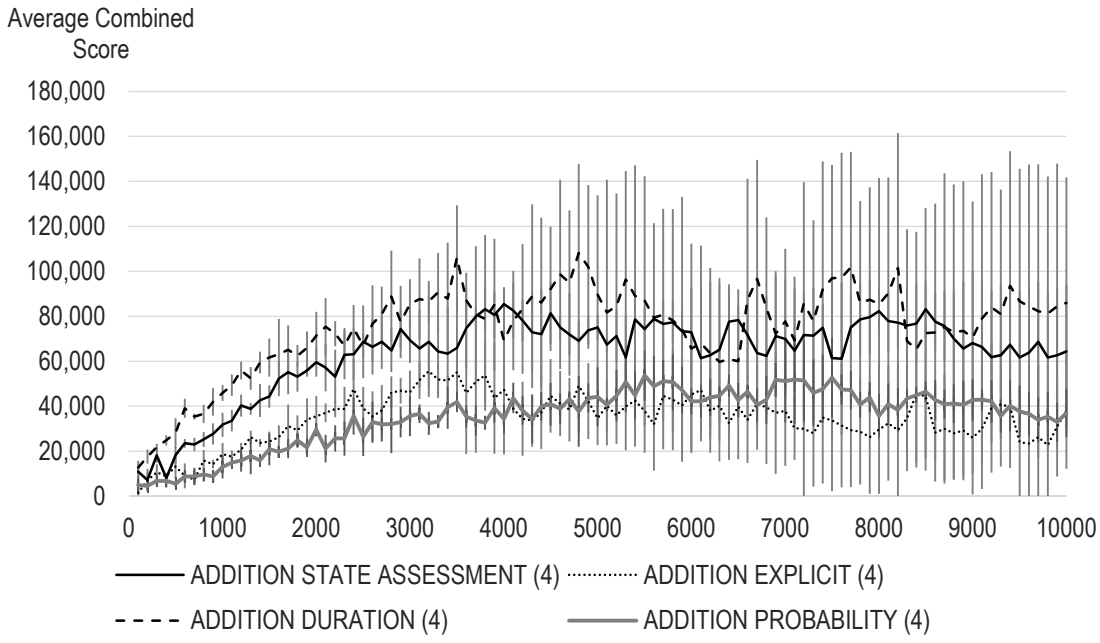

Figure 8.5 Termination Criteria and Convergence

Overall, agents that executed options for longer durations (either via state assessment or explicit durations) fared better than those who only executed those options either explicitly or based on some probability. In no case, though, were two agents successful in retrieving both flags. Four cases were compared for each type of run.

Figure 8.6 illustrates the differences in option duration based on the termination criteria employed. Much like the learning outcomes, using a termination strategy of either assessing the current state to see if the options were being followed in spirit, if not fact or encouraging the options to execute for longer durations resulted in longer durations of option execution.

Adjusting the termination criteria improved overall performance, but did not have the desired outcome of encouraging the agents to converge to a jointly optimal solution. On further analysis, this could be due to the allocation of future utility to agents, thereby incentivizing agents to participate in options when there truly is nothing in it for them.

8.4 Summary

In this chapter, an assessment of the challenges associated with using the reduced format game in a distributed action selection environment was presented. To counter the situation where agents selected actions which were invalid partitions of the agent space, the concept of placeholder

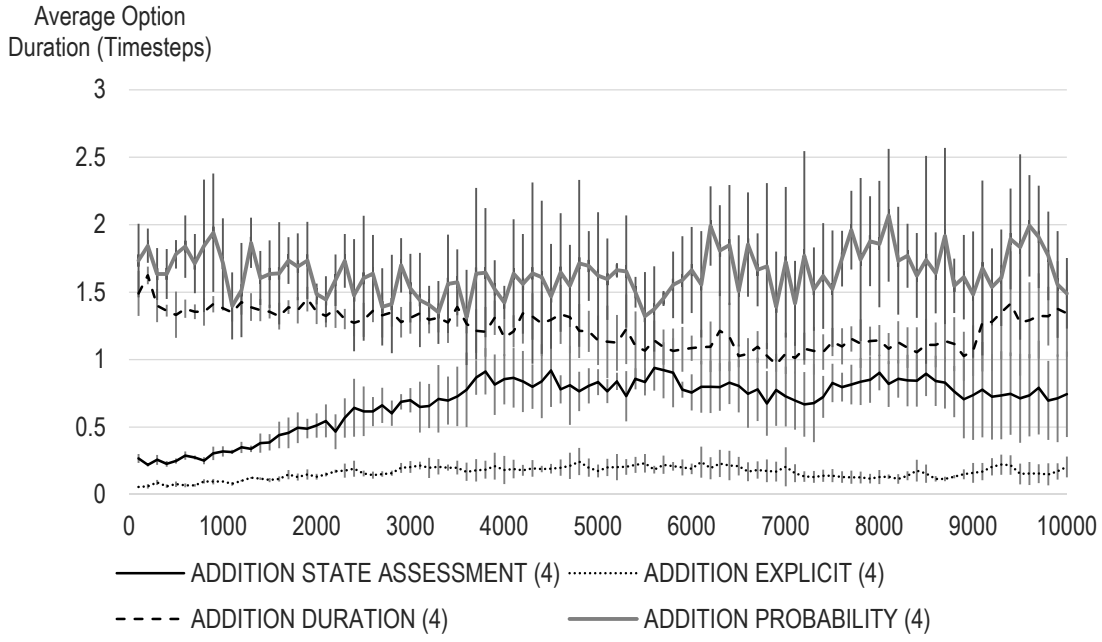

Figure 8.6 Termination Criteria and Average Option Duration

Both state assessment and forced duration of option execution (unsurprisingly) result in longer average option durations.

options was presented. In experiments run to support the distributed scenario, these options did aid in the learning updates for agents, but the agents did not converge to the optimal policy. This is a common issue with employing these types of learning algorithms in a distributed environment. To address this, some potential further updates will be presented in the next chapter.

CHAPTER 9

Conclusions and Future Work

The research presented in this dissertation focused on reasoning about the world in terms of sets of agents, not individual entities. It is a unique approach in the current body of research, and as such has laid a foundation for future refinements and enhancements.

9.1 Conclusions

This dissertation presented research on a method to evaluate multiagent learning systems as sets of agents. This method allows agents to reason about subsets of their social world. In early chapters, this document defined a **game-theoretic approach to reducing system dimensionality through multiagent options.** These options were applied and evaluated in two-player scenarios to provide experimental evidence of reduced learning time and policy convergence. Methods for equilibria evaluation and termination approaches were also assessed in the two-player domain.

After exploring the behaviors of agents within a coalition, that approach was extended to **define how sets of agents interact with each other.** Coalitions, bound options and roles were all defined in support of the multi-agent-set domain. The reduced format game was defined and presented as a mechanism to reason about sets of other agents in a lower-dimensionality state space.

An approach to combine agent values was also presented. Equilibria calculated from games using these combination techniques were also shown to be equilibria in the full game.

Finally, experiments on exploiting this structure in a distributed game were also presented. The concept of placeholder options was introduced and explored. This exploration presented new opportunities for further research on how to maintain coalition utility.

9.2 Future Work

There are many future refinements that can enhance the results presented in this dissertation. Some of those include: refinements to support convergence in the distributed case, assessment of stability and extensions to the initiation sets, and applying these algorithms hierarchically.

9.2.1 Convergence and Exploration Refinement

The most immediate next step for this research is to assess means to encourage convergence in the distributed case. Some possible approaches to doing so include: adapting the exploration from an ε-greedy approach to a more targeted exploration mechanism, allowing agents to "join" multiagent options already in progress, and providing early opportunities for collaborative option selection.

9.2.2 Stability and Extensions to the Initiation Set

Other refinements to this research could include the extension of initiation set logic to look at the stability and fairness of coalition behaviors. Multiagent options are, by definition, internally equilibria, but when operating in the context of additional coalitions, there may be points at which agents would never elect to participate in those options. In those cases, the options themselves could be removed from consideration in the reduced format game.

9.2.3 Scaling to Larger Numbers of Agents

This formulation will show significant benefit if it can be applied hierarchically to domains of more and more agents. If single agent options could be learned which then comprise multiagent options for two players, which then comprise multiagent options for three and four players, the system could scale such that an agent might never have to reason about more than two entities. This structure could significantly reduce learning time for the overall system, but implementation and experimentation at that scale requires the implementation of some amount of state abstraction capability into the system.

9.3 Summary

This dissertation presented a multiagent approach to addressing the curse of dimensionality in terms of agent set abstractions, leveraging the concept of multiagent options to define behaviors for sets of agents. Reduced learning time when using these multiagent options was demonstrated. When considering interactions between coalitions, this research defined a new game structure – the reduced format game – which allowed agents to reason about sets of agents as a single entity. Correspondence of equilibria between the reduced format game was shown. A mechanism to calculate and maintain solely the utilities of coalitions was explored, and while some refinements would enhance its performance – especially in the distributed case – it also was demonstrated to converge to an optimal solution. When structuring the reduced format case, the concept of placeholder options was also introduced, as a mechanism to capture value when not all players select the same activity. Although there is much future work to be done in the distributed case, the mechanisms presented lay a strong groundwork for those future excursions.

APPENDIX A

GAMBIT Configuration and Performance Characterization

The game theory engine selected to be used to compute equilibria for this research is GAMBIT[43]. Established in the 1990s, since 2011 GAMBIT has been maintained and extended as a part of the Google Summer of Code [44]. GAMBIT is a software package that provides implementations of many different solving algorithms to compute Nash equilibria, as well as a user interface to visualize and generate the games. The portion of the GAMBIT suite relevant to this research is the set of command line utilities that provide different solving algorithms to compute Nash equilibria. Other game theory solvers are available, but they are predominantly limited in the size of the games they can compute or tailored to specific use cases (such as web-based game solving) [45]–[47], or already incorporated into GAMBIT[48]. For this research in particular, GAMBIT is the most suitable tool to generate equilibrium solutions.

GAMBIT has been referenced in nearly 200 publications over the past 10 years. The research described in this dissertation calculates Nash Equilibria twice (once for the action choice, once for the Nash update) with every non-exploration time-step when executing NashQ and once (for the

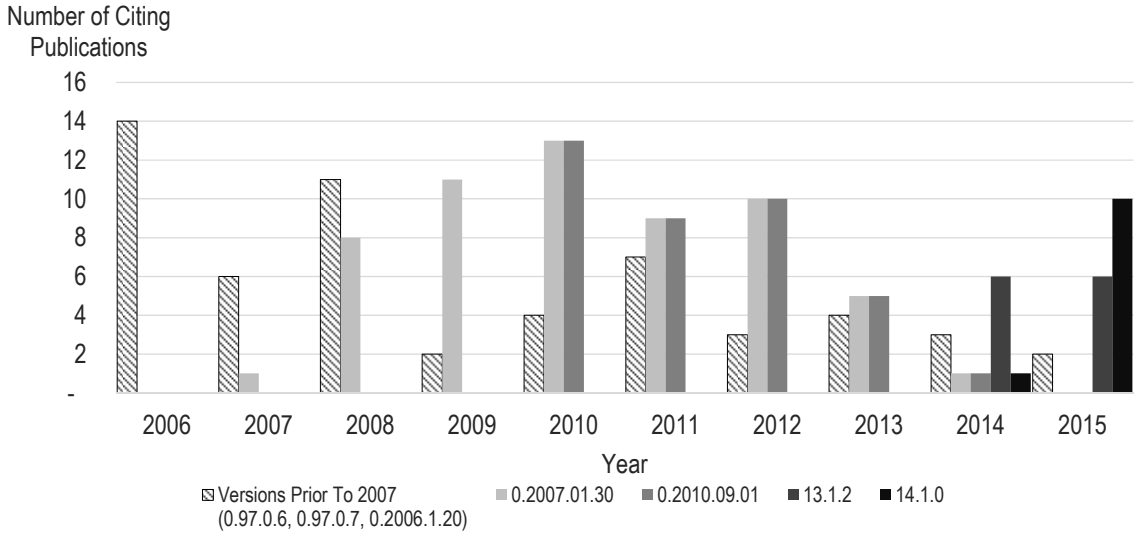

Figure A.1 Papers Citing GAMBIT 2006-2015

In the past 10 years, a search of papers citing GAMBIT by version reveals the illustrated usage patterns. GAMBIT version prior to 2007 are bundled into a single data point. Those versions are still in use today. The overall most-cited GAMBIT version is 0.2007.01.30, with 0.2010.09.01 following. In more recent years, publications more frequently cite the 14.1.0 version. A 15.0.0 version also exists but has not yet been cited.

action choice) when executing Nash SARSA. This can be over one million calls per agent in a single evaluation experiment, so it is critical when computing equilibria that the solving algorithm be fast, accurate and robust.

The remainder of this appendix describes performance testing that the author used to characterize the behavior of select GAMBIT command line utilities in order to better incorporate those utilities in the source code baseline used for the experiments described in the body of this dissertation. Recommendations to other users of the GAMBIT tool suite are provided at the completion of this appendix.

A.1 Experimental Description

A.1.1 GAMBIT Command-Line Solvers

For this evaluation, we considered several of the solvers provided by the GAMBIT tool suite, namely enummixed, gnm, liap, simpdiv, and lcp. Of these solving algorithms enummixed and lcp only work on two player games. Further details on these algorithms, including the citations for their detailed descriptions, are provided in the existing GAMBIT online documentation. Other solvers are provided by GAMBIT but were not assessed here.

These evaluations were completed to characterize performance for the implementation associated with this research. These results are not intended to assess the complexity or best possible run-time of these algorithms, only of their implementation as released in these described versions of GAMBIT and as executed in the runtime environment described below (which is consistent with the runtime environment used for this research).

A.1.2 Evaluation Environment

The experiments described in the following sections were executed on an Intel i7-4770 processor at 3.4GHz (quad-core) with 8GB RAM running 64-bit Ubuntu 14.04. This run-time environment is comparable to the environment used to generate other experimental data presented in this dissertation, with the exception of installed RAM. RAM was not a limiting factor in the execution of

these experiments. The GAMBIT executables were built and installed per the provided instructions and makefiles, with –disable-gui and –no-enumpoly flags set. The disable gui setting builds only the command line execution tools, and the enum poly solver is incompatible with 64-bit machines, and therefore disabled.

To assess the performance of different solving algorithms, solver execution was timed against randomly generated 2 player games. Payoffs were generated randomly in the range of (-9,9). Each solving algorithm was assessed independently against each game. The action space ranged from 2 to 50 actions per player, incremented by 3 actions each step (so 2x2, 2x5, 5x5, etc games were all generated). For each permutation of the action space, 25 games were generated. To support the data analysis that is shown in the following charts, games were allocated to analysis groups based on total state space size in intervals of 5.

In some cases, the solving algorithms do not return control to the parent process. To allow for continuation of the assessment, a timeout was implemented. If the solver attempts to execute for longer than this timeout, the solver process will be killed. Even if a timeout occurs, however, some equilibria may be returned. In addition, there are also cases where the solver finds no equilibria. In many of those cases, the solver will return cleanly to the parent process. Assessment of the number of occasions where no equilibria are returned is also presented in this analysis.

Relevant game state sizes for the reduced form game experiments presented in this dissertation range from 140-440 states. Those ranges of states are highlighted in the following graphs to provide context for the selections made.

1.1.3 Metrics Captured

Calculation Time

Because this research uses NashQ and Nash SARSA to generate and update multiagent policies, games are solved repeatedly. The time required to calculate an equilibrium solution is the key factor in determining overall research solution time. For this reason, a major concern to this research is finding an equilibrium solver that calculates correct equilibria in a minimum amount of time. To

assess the solution time of these algorithms, the system time elapsed between the parent process requesting a solution and the command line solver returning control or timing out was captured. Any post-processing of the returned equilibria, including assessment of equilibria for validity and the time required to capture and report of metrics were not included in the timing assessment.

Percentage of Time Returning a Valid Equilibria

In the course of this evaluation, it became apparent that the GAMBIT command-line equilibrium solvers occasionally return solutions that are not valid equilibria. An example 2x2 case where an invalid equilibria is returned from some GAMBIT command line solvers is shown in Figure A.2. In this game, there is an equilibrium solution at {Y,B}. The payoffs for this solution are {-1, -1}. Some of the GAMBIT command-line solvers also return the solution {X:.78, Y:.22, B:1}. The payoffs for this solution are {-1, 2.90}. However, when player 2 deviates to action A, Player 2's payoff for this solution is 2.92 (which is an improvement over the payoff of 2.90). Therefore, action B is not a best response to Player 1's strategy and this solution is not an equilibrium. This particular case appears to be a result of calling the equilibrium solver using the "-d" flag. The "-d" flag, combined with an integer (here using 2) indicates to the solver that a decimal representation of the answer should be returned. When that flag is omitted, the equilibrium returned is {X:7/9, Y:2/9, B:1}, which results in a payoff to Player 2 of 2.888 for following the equilibrium strategy and alsofor deviating (thereby making this a weak equilibrium). In this assessment, we provide a comparison of valid equilibria

	A	B
X	-8, 6	-1, 4
Y	5, -8	-1, -1

Figure A.2 An Example Matrix Game Which Generates an Invalid Solution

This randomly generated 2x2 matrix game returns an invalid equilibrium solution when evaluated by the GAMBIT command line solvers ENUM_MIXED, LCP and LIAP with the "-d" decimal flag. Valid equilibria are returned for this particular game when the decimal flag is excluded and returned values are parsed as strings.

returned when parsing a string representation of the equilibria vs. a decimal representation. After this discovery, when using GAMBIT in this research, all equilibria returned from the GAMBIT were checked for validity before being made available to the rest of the system. Equilibria deemed invalid (where the payoff for at least one agent when deviating from the equilibrium strategy was strictly greater than the payoff for playing the equilibrium strategy) were not returned to the parent process. In some cases, especially for higher-dimension games, this validity check eliminated all of the potential returned solutions.

Percentage of Games Where At Least One Returned Equilibria Was Valid

While the overall percentage of valid equilibria is interesting from a processing perspective, to be successfully employed in the software used for this research, an algorithm only needs to return a single equilibria. The number of games within a given state space where at least one valid equilibria were returned was calculated and divided by the total number of games in that state space to compute this metric.

In some cases, the GAMBIT algorithms return with no identified solution, or, as described above, sometimes the solutions which are returned are all invalid and are all eliminated. For this reason, in this research a stack of solving algorithms was implemented, so that if one solver returned no solution, a different solver could attack the problem. This approach resulted in equilibria returned to the calling algorithms almost all of the time. However, in cases where it did not, the last selected equilibrium (saved for every state) was returned. If there was no previous solution, the software developed to support this research either selected a random course of action for agents (if this occurred during the action selection phase) or did not update Q-values (if this was in the update phase for NashQ).

Percentage of Time Outs

In a few instances, some of the solving algorithms implemented by GAMBIT will not return control to the parent process. For this reason, a timeout was implemented (in conjunction with a non-blocking read) in the parent process. This timeout allowed the calling function to select a new solving algorithm to execute if the original algorithm did not return any found equilibria. For this

evaluation, the number of instances the solving algorithm timed out was captured. Note that this does not necessarily mean that the solving algorithm returned no equilibria.

Total Number of Equilibria Returned

While it is critical to this research that at least one equilibrium solution is returned, processing and validating each returned equilibria also carries a cost. When significantly more than one equilibrium is returned, this cost can become prohibitive. For this reason, the total number of equilibria returned is also evaluated.

A.2 Game Engine Evaluation (Version 13.1.2)

A.2.1 Calculation Time

One of the primary concerns to this research in regard to equilibrium solvers is the speed at which a solution is obtained. The demonstrated solution time varies by solving algorithm with the scale of the problem under consideration, where the scale of the problem is defined as the number of states in the game. As shown in Figure A.3, for the range of 140 to 440 states, only the GNM and SIMPDIV algorithms return consistently before the timeout. Of these, the ENUM MIXED algorithm performs the worst, as it times out in games that cover nearly the entire relevant state space.

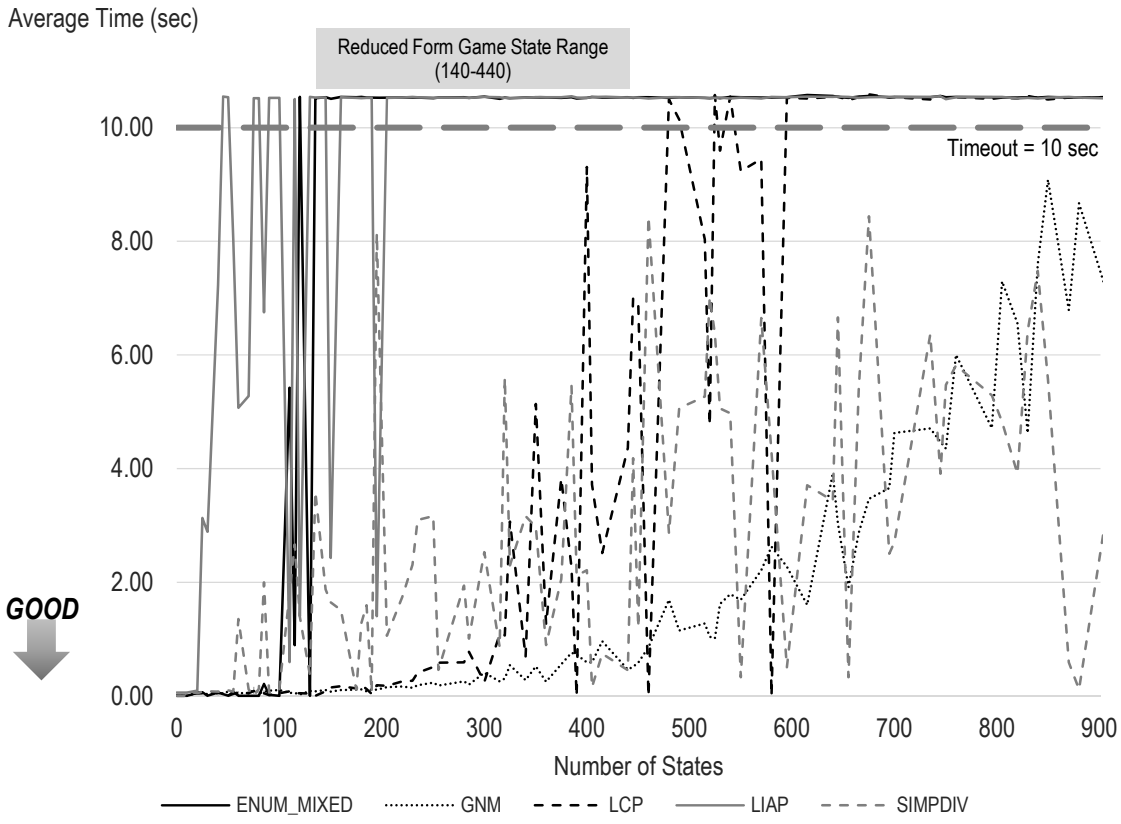

Figure A.3 Time to Calculate Solution (Including When No Equilibria Are Returned)

In the relevant state range, the SIMPDIV and GNM algorithms consistently return solutions without timing out. While useful in games with smaller state space sizes, the LCP, LIAP and ENUM-MIXED algorithms time out consistently at some number of relevant game states (given a timeout setting of 10 seconds).

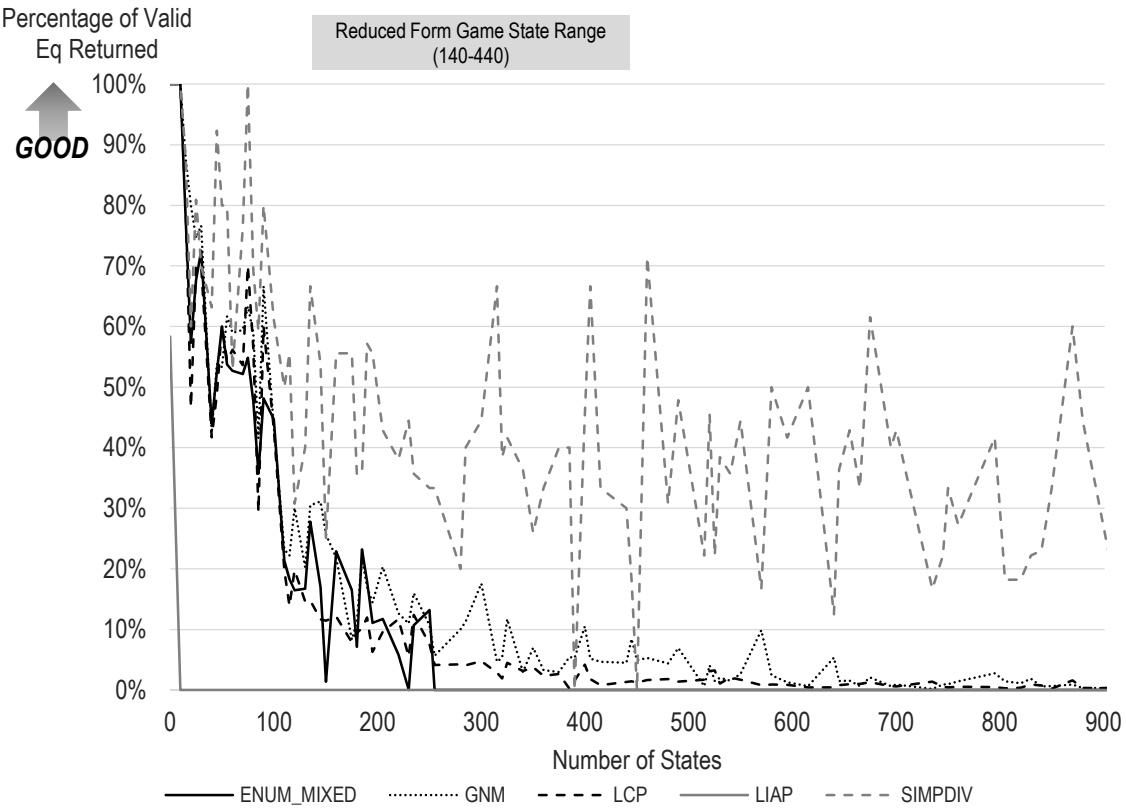

Figure A.4 Percent of Equilibria Returned Which Were Valid

Performance in returning valid equilibria in the relevant reduced form game space is disappointing. All equilibria returned by the SIMPDIV algorithm are valid. When faced with greater than 120 game states, however, ENUM_MIXED fails to return valid solutions. LIAP fails at state spaces greater than 10. GNM and LCP will both return some valid equilibria in the relevant range.

A.2.2 Percentage of Time Returning a Valid Equilibrium

Each solution returned was evaluated against the game to ensure that no player could benefit by deviating from the equilibrium strategy. This evaluation revealed that LIAP does not return any valid equilibria after the game state size exceeds 10. ENUM MIXED stops returning valid solutions after 120 game states. SIMPDIV, however, will always return a valid solution. The performance of GNM and LCP are minimally successful in the period of the relevant game state space sizes.

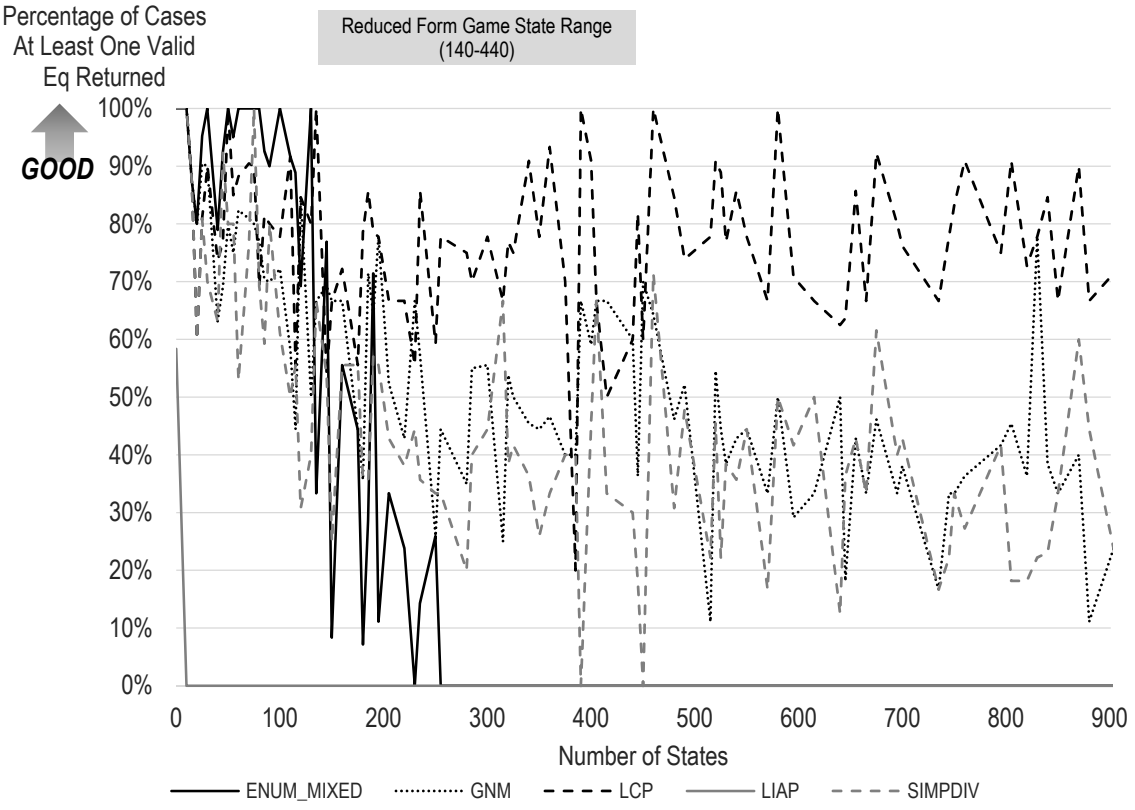

Figure A.5 Percent of Games Where At Least One Equilibrium Was Valid

Overall, LCP is the algorithm most likely to return at least one valid equilibrium. Performance of ENUM MIXED and LIAP were especially poor, with LIAP only returning valid solutions in very small state spaces, and ENUM MIXED only valid until around 250 states. Performance of GNM and SIMPDIV are mixed, but equilibria are returned frequently.

A.2.3 Percentage of Games Where At Least One Returned Equilibria Was Valid

Based on the assessment in the previous section, the number of games where at least one of the equilibria returned was calculated. The percentage overall of those games is displayed in Figure A.5. ENUM MIXED and LIAP performed poorly, with the former timing out after less than 10 states and the latter only returning valid equilibria at less than 250 states. Performance of all other algorithms was mixed, with LCP providing valid equilibria the most frequently in higher dimension games.

A.2.4 Percentage of Time Outs

These algorithms were evaluated against a timeout of 10 seconds, meaning that the process would be terminated after 10 seconds had elapsed. This is critical regardless of whether or not the algorithm returns solutions, because longer solving times extend the time required for the software to execute. The algorithms can be called over a million times in a single test run, so if each of those calls resulted in a timeout, test runs would take over 115 days to complete. If the solver has found a solution and is looking for more, it is to the benefit of the calling software (for this application) to halt it's execution. Conversely, if the solver has found no equilibria in that time period, it is to the benefit of the calling software to determine another approach. As shown in Figure A.6, LIAP and ENUM MIXED consistently time out in games with more than 100 game states. The remainder of the algorithms time out less than 50% of the time in the relevant range of game states.

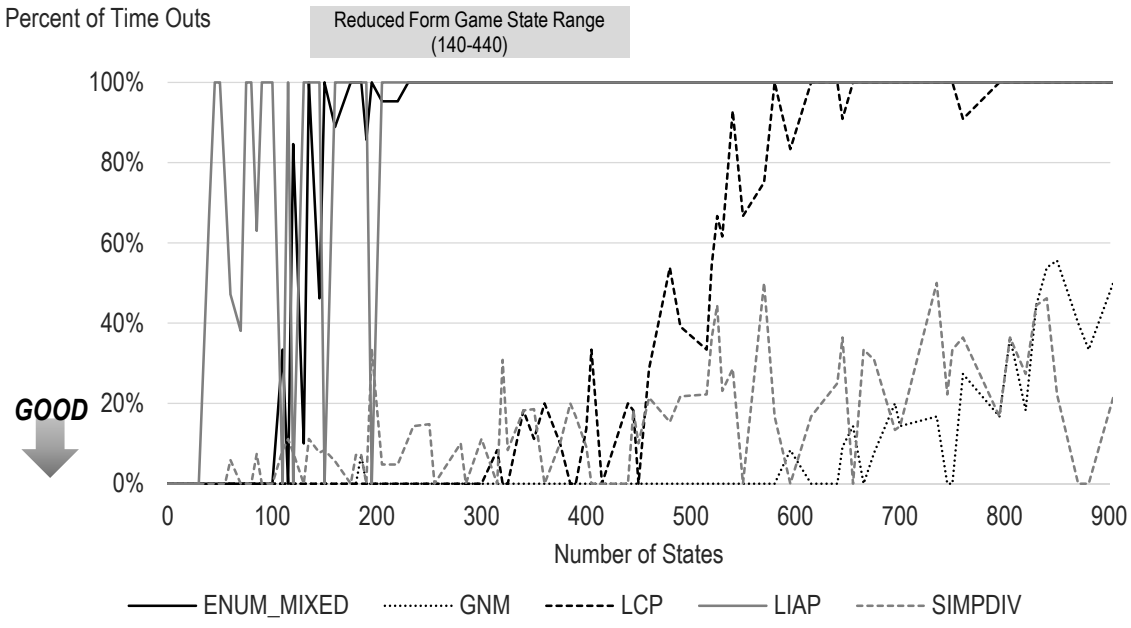

Figure A.6 Solving Algorithm Timeouts

LIAP and ENUM MIXED consistently time out once the number of game states reaches 200. The rest of the algorithms perform well, only rarely timing out in the range of the reduced form game.

A.2.5 Total Number of Equilibria Returned

In a few instances, some of the solving algorithms implemented by GAMBIT will not return control to the parent process. For this reason, a timeout was implemented (in conjunction with a non-blocking read) in the parent process. This timeout allowed the calling function to select a new solving algorithm to execute if the original algorithm did not return any found equilibria. In Figure A.7, of note is that all the solvers return different numbers of equilibria for different games, and of those solvers, LCP appears to return the most equilibria (on average).

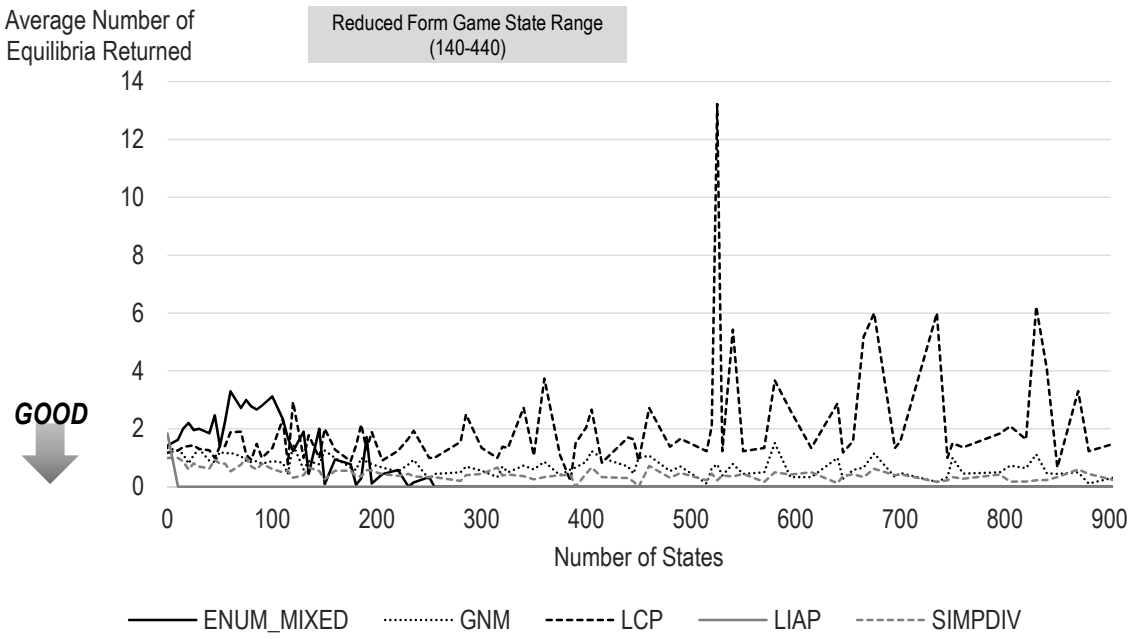

Figure A.7 Average Number of Equilibria Returned

Of note here is that the solving algorithms, operating on the same games, return different numbers of equilibria. In addition, the LCP solver appears to return the most equilibria in higher dimension state spaces.

A.2.6 String Equilibria vs. Decimal Equilibria

After assessing the number of valid equilibria returned and comparing those results between the string and the floating point calculations, with the exception of some very small game state sizes, there is no significant difference between the number of valid equilibria returned when using the string or decimal representation.

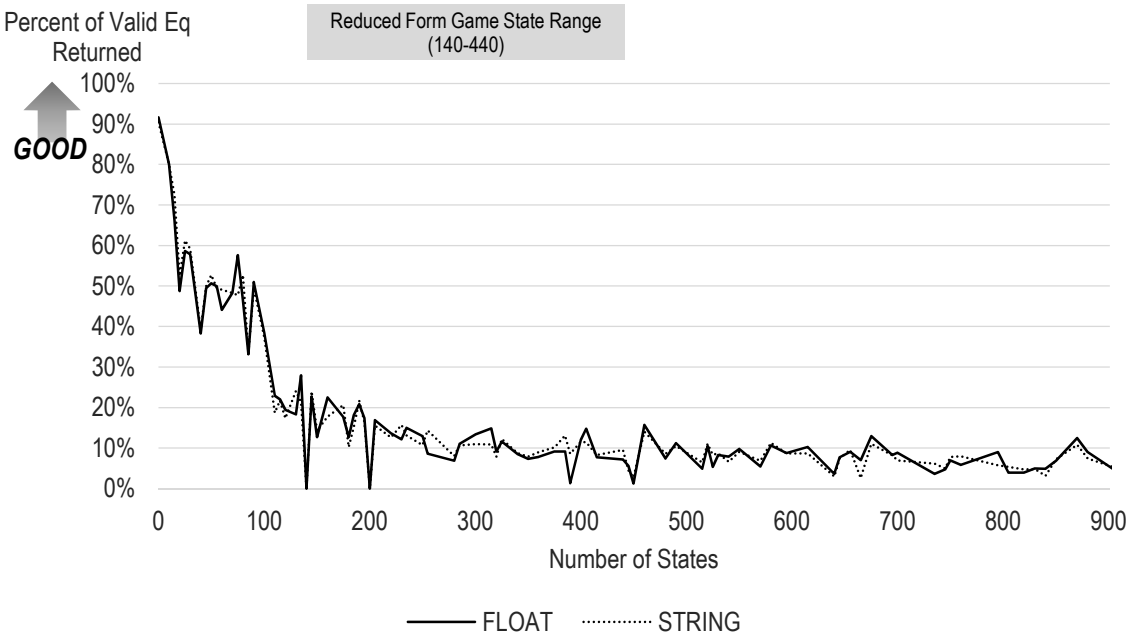

Figure A.8 Number of Valid Equilibria Returned Using String or Decimal Output Method

The difference in number of valid equilibria returned when using the "-d" GAMBIT flag appears to be very small, with both methods returning a fairly low percentage (10% or less) of valid equilibria once games exceed 200 states.

Table A.1 Summary of Performance of GAMBIT 13.1.2 Solving Algorithms

This table summarizes the performance of the evaluating GAMBIT equilibria solving algorithms, when scored against randomly generated two player games with payoffs in the range of (-9, 9 with 10 second timeout).

Solving Algorithm	***Calculation Time***	***Percentage of Valid Equilibria Returned***	***Percentage of Time Returning a Valid Solution***	***Percentage of Timeouts***	***Total Number of Equilibria Returned***
ENUM MIXED	Timeout consistently in games with more than 150 states	Less than 10% of returned equilibria were valid in games with more than 200 states	Returned no valid solution in games with more than 250 states	Times out consistently in games with more than 100 states.	Consistent performance with the other algorithms, until game state size reaches 250.
GNM	Does not consistently timeout in the relevant range	Less than 10% of returned equilibria were valid in games with more than 350 states	Sporadically returned valid equilibria.	Does not consistently time out in the relevant range.	Consistent performance with the other algorithms.
LCP	Times out in games with more than 600 states	Less than 10% of returned equilibria were valid in games with more than 250 states	Best performer. Returned at least one valid equilibrium 80% of the time (on average)	Times out consistently in games with more than 100 states.	Sometimes more than twice as many equilibria are returned as compared to the other algorithms.
LIAP	Timeout consistently in games with more than 50 states	Returned no valid equilibria for games with more than 10 states	Never returned a valid solution in games of more than 10 states.	Times out consistently in games with more than 600 states.	Never returned a valid solution in games of more than 10 states.
SIMPDIV	Does not consistently timeout in the relevant range	Best Performer. Returned valid equilibria 50% of the time (on average)	Sporadically returned valid equilibria.	Does not consistently time out in the relevant range.	Consistent performance with the other algorithms.

A.2.7 Summary of Results

The qualitative performance of the solving algorithms is summarized in A.1.2. Based on the assessment summarized in Table A.1, the software used to support this dissertation uses first the GNM algorithm, with fall backs of LCP (for 2 player games), ENUMMIXED (for two player games) and SIMPDIV algorithms should no equilibria be returned before the time out occurs.

A.3 Assessment of Representative Games

While the randomly generated games provide an interesting baseline for this assessment, the motivation for performing this analysis was to assess the probability of success of incorporating the

GAMBIT solvers into the code required for this research – and the games used in this research are very structured. Because of this structure, there are significant numbers of dominated strategies that could be eliminated prior to solving these games. A sample 30 games from the final episodes of one of the “sum” experiments described in Section 7.4 were culled and assessed to determine if the solver performance varied with respect to these more representative games.

In general, the representative games were significantly more likely to return equilibria before timing out. Of interest, however, was the performance of the ENUM_MIXED algorithm on these games. This algorithm would return orders of magnitude more equilibria than the other algorithms against the same games. While the calculation itself typically did not timeout, the sheer volume of equilibria resulted in long post-processing times (which are not reflected in this analysis). For this research, a single consistently returned equilibrium is sufficient, so consistently returning 4800 spare choices results in overall system slowdown.

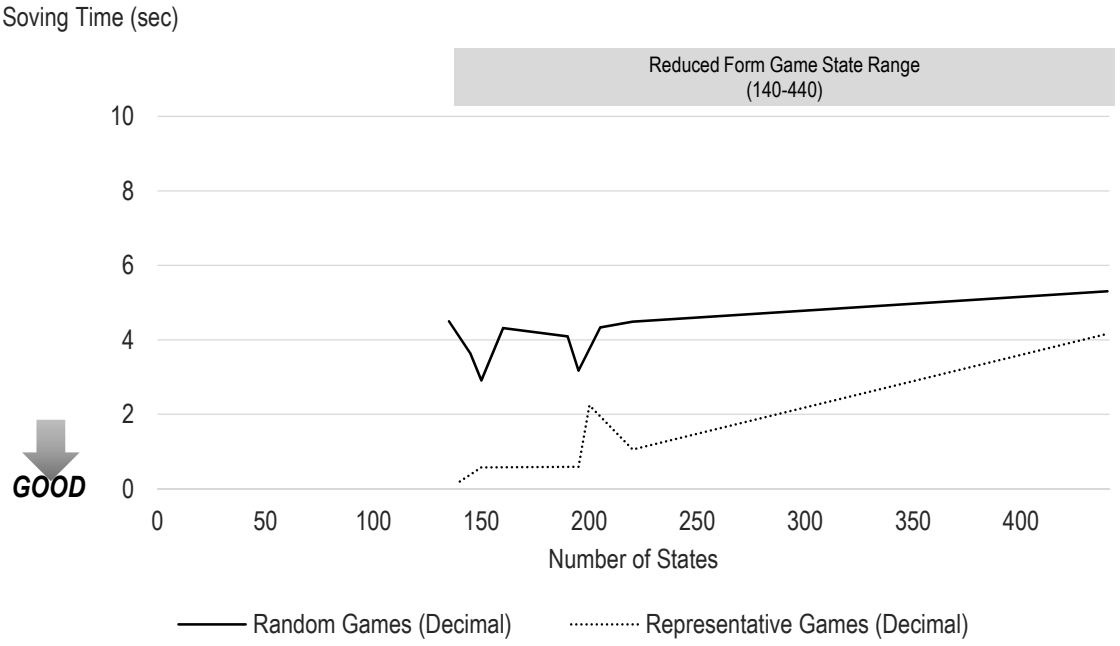

Figure A.9 Time to Calculate Solution (Including When No Equilibria Are Returned)

The representative games consistently return in less time than the randomly generated games.

A.3.1 Calculation Time

As shown in Figure A.9, the solving algorithms on representative games require significantly less time to return an answer than the randomly generated games over all solving types.

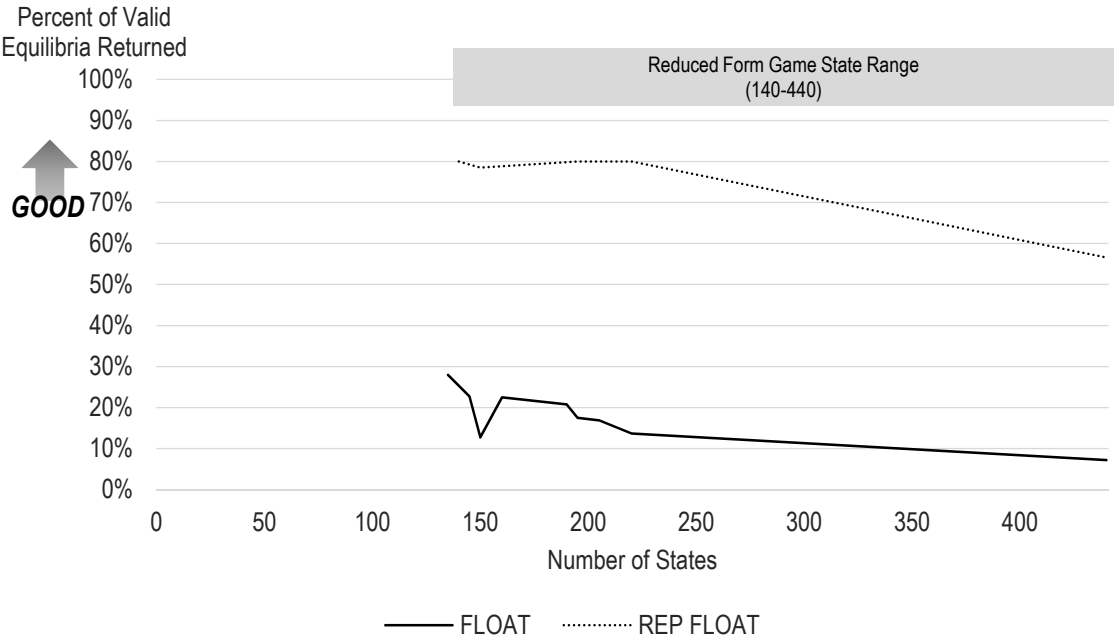

Figure A.10 Percent of Time Returning a Valid Equilibrium

Over the range of representative games, the solvers consistently perform with fewer timeouts against the representative games as opposed to the random games.

A.3.2 Percentage of Time Returning a Valid Equilibrium

Solver performance is also significantly improved against the representative games in terms of returning valid equilibria, as shown in Figure A.10.

A.3.3 Percentage of Games Where At Least One Returned Equilibria Was Valid

As illustrated in Figure A.11, the frequency with which the algorithm returned without providing a solution was significantly improved for the representative games as opposed to the randomly generated games.

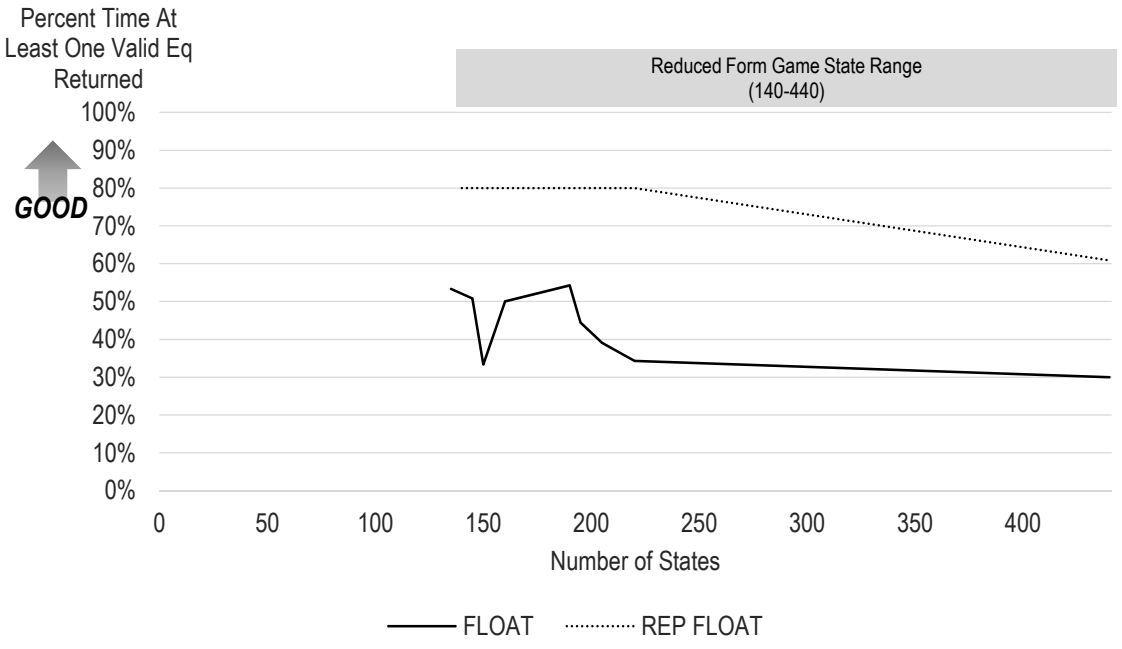

Figure A.11 Percent of Time Returning at Least One Valid Equilibria

The solving algorithms overall returned at least one solution almost 80% of the time against the representative games, a significant improvement over the randomly generated games.

A.3.4 Percentage of Time Outs

The representative games also caused solving algorithms to time out significantly less than the randomly generated algorithms, as shown in Figure A.12.

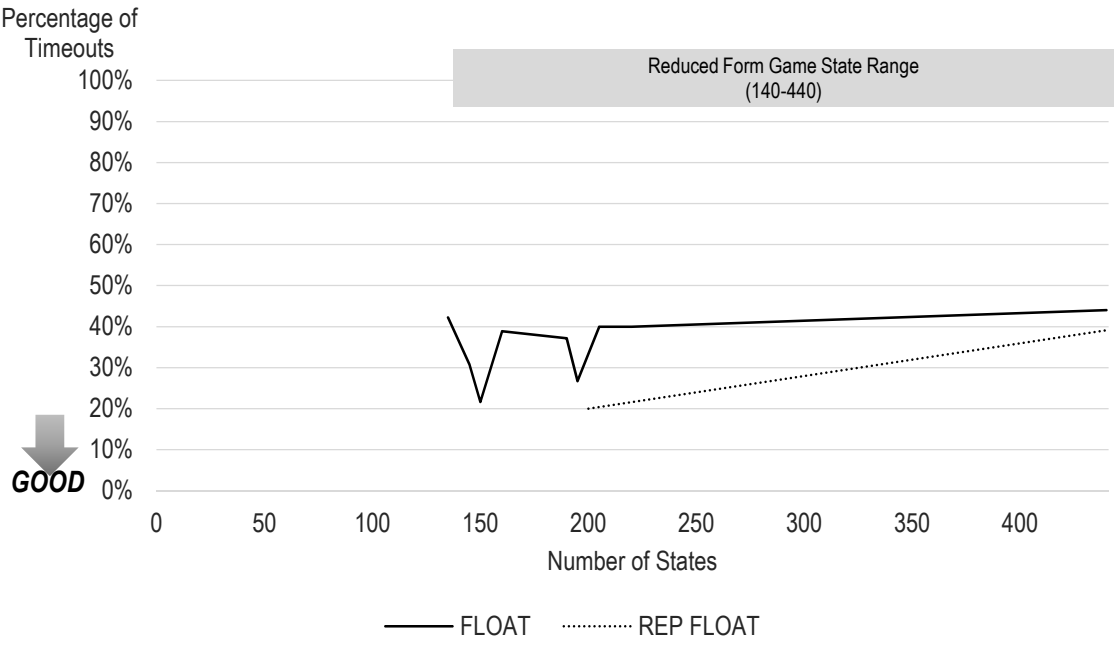

Figure A.12 Solving Algorithm Timeouts

The representative games also resulted in timeouts significantly less frequently than the randomly generated games.

A.3.5 Total Number of Equilibria Returned

Against the representative games, a significantly greater number of equilibria were returned, as depicted in Figure A.13. This significant increase was primarily due to the ENUM MIXED algorithm. Although overall solution times improved against the representative games, the associated validity checks and ranking algorithms (not timed) used by the software operate on all valid returned equilibria, making this surplus time-consuming in the post-processing of these solutions.

A.3.6 Summary of Results

Overall, the representative games were managed more efficiently by the solving algorithms. The significant number of equilibria returned by the ENUM MIXED algorithm is of interest when generating the stack of solvers.

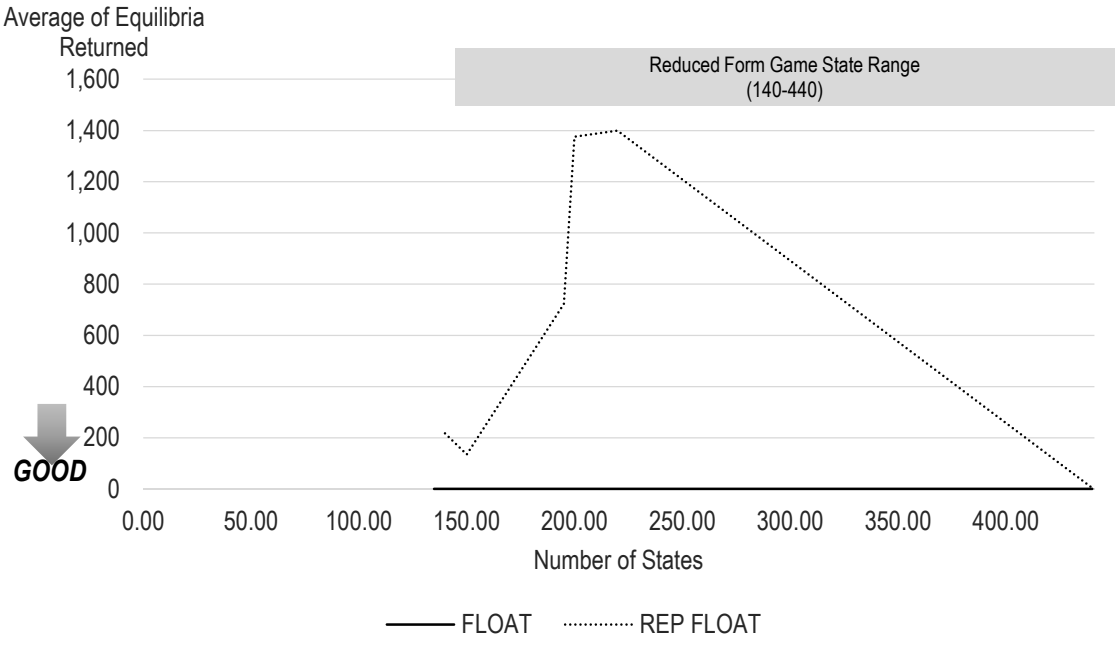

Figure A.13 Total Number of Equilibria Returned

The total number of equilibria returned (on average) in the representative games was significantly greater than the number returned from the randomly generated games.

A.4 Summary and Recommendations to Other Users

GAMBIT is a valuable tool to compute equilibria solutions for games. It allows multiagent reinforcement learning researchers and developers to quickly integrate game theoretic solution approaches into their software. However, considered selection of solver methods must occur, especially in regard to the number of equilibria returned and the computation time required to reach a solution. In addition, non-blocking reads should be employed with timeouts to ensure that if the software does not return expeditiously, the calling program can respond. Similarly, it is possible for an individual solving algorithm to return no equilibria, so maintaining a stack of potential solvers and calling the next in the stack should one fail helps to ensure that solutions are eventually reached. Finally, equilibria returned should be checked for validity before being used, especially in higher-dimensional state spaces and when using the decimal representation flag, as some of the equilibria returned are not valid solutions in the game.

REFERENCES

[1] R. Bellman, *Adaptive Control Processes: a guided tour.* Princeton, NJ, 1961.

[2] S. Haslam, “Stereotyping and social influence: Foundations of stereotype consensus.,” 1997.

[3] J. Turner and P. Oakes, “The significance of the social identity concept for social psychology with reference to individualism, interactionism and social influence,” *Br. J. Soc. Psychol.*, vol. 25, no. 3, 1986.

[4] A. Barto and R. Sutton, *Reinforcement learning: An introduction*. 1998.

[5] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, Second Edi. Englewood Cliffs: Prentice-Hall, 2003.

[6] C. Watkins, “Learning from delayed rewards.,” University of Cambridge, 1989.

[7] G. A. Rummery and M. Niranjan, “On-Line Q-Learning Using Connectionist Systems,” 1994.

[8] R. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artif. Intell.*, vol. 112, no. 1, pp. 181–211, Aug. 1999.

[9] S. J. Bradtke and M. O. Duff, “Reinforcement learning methods for continuous-time Markov decision problems,” *Adv. Neural Inf. Process. Syst.*, vol. 7, no. 7, p. 393, 1995.

[10] J. Provost, B. Kuipers, and R. Miikkulainen, “Self-organizing perceptual and temporal abstraction for robot reinforcement learning,” in *AAAI Workshop on Learning and Planning in Markov Processes*, 2004, pp. 79–84.

[11] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. 2008.

[12] J. Nash, “Non-cooperative games.,” *Ann. Math.*, pp. 286–295, 1951.

[13] J. Hu and M. P. Wellman, “Nash Q-learning for general-sum stochastic games,” *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, 2003.

[14] P. Stone, R. S. Sutton, and G. Kuhlmann, “Reinforcement Learning for RoboCup-Soccer Keepaway,” *Adapt. Behav.*, vol. 13, no. 3, pp. 165–188, 2005.

[15] L. Panait and S. Luke, “Cooperative Multi-Agent Learning: The State of the Art,” *Auton. Agent. Multi. Agent. Syst.*, vol. 11, no. 3, pp. 387–434, Nov. 2005.

[16] T. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.

[17] R. Parr and S. Russell, “Reinforcement learning with hierarchies of machines,” in *Advances in neural information processing systems*, 1998, pp. 1043–1049.

[18] Y. Cai, S. Yang, and X. Xu, “A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments,” in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, 2013, pp. 52–59.

[19] R. Makar, S. Mahadevan, and M. Ghavamzadeh, “Hierarchical multi-agent reinforcement learning,” in *Proceedings of the fifth international conference on Autonomous agents*, 2001, pp. 246–253.

[20] J. Wu, X. Xu, P. Zhang, and C. Liu, “A novel multi-agent reinforcement learning approach for job scheduling in Grid computing,” *Futur. Gener. Comput. Syst.*, vol. 27, no. 5, pp.

430–439, May 2011.

[21] H. Kawano, “Hierarchical sub-task decomposition for reinforcement learning of multi-robot delivery mission,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 828–835.

[22] M. Leece and A. Jhala, “Reinforcement Learning for Spatial Reasoning in Strategy Games,” in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013, pp. 156–162.

[23] M. Hawasly and S. Ramamoorthy, “Lifelong learning of structure in the space of policies,” *AAAI Spring Symp. Lifelong Mach. Learn.*, pp. 21–26, 2013.

[24] K. Rohanimanesh and S. Mahadevan, “Learning to take concurrent actions,” in *Advances in Neural Information Processing Systems*, 2002, pp. 1619–1626.

[25] K. Rohanimanesh and S. Mahadevan, “Decision-theoretic planning with concurrent temporally extended actions,” in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 2001, pp. 472–479.

[26] W. Cao, G. Chen, X. Chen, and M. Wu, “Optimal tracking agent: a new framework of reinforcement learning for multiagent systems,” *Concurr. Comput. Pract. Exp.*, vol. 25, no. 14, pp. 2002–2015, 2013.

[27] N. Aissani, A. Bekrar, D. Trentesaux, and B. Beldjilali, “Dynamic scheduling for multi-site companies: a decisional approach based on reinforcement multi-agent learning,” *J. Intell. Manuf.*, vol. 23, no. 6, pp. 2513–2529, Aug. 2012.

[28] C. HolmesParker, A. Agogino, and K. Tumer, “Combining reward shaping and hierarchies for scaling to large multiagent systems,” *Knowl. Eng. Rev.*, pp. 1–18, 2013.

[29] C. Wernz and A. Deshmukh, “Unifying temporal and organizational scales in multiscale decision-making,” *Eur. J. Oper. Res.*, vol. 223, no. 3, pp. 739–751, Dec. 2012.

[30] C. Wernz, “Multi-time-scale Markov decision processes for organizational decision-making,” *EURO J. Decis. Process.*, vol. 1, no. 3–4, pp. 299–324, Nov. 2013.

[31] C. Amato, G. Konidaris, and G. Cruz, “Planning for Decentralized Control of Multiple Robots Under Uncertainty,” in *ICRA*, 2014, pp. 1241–1248.

[32] C. Zhang, S. Abdallah, and V. Lesser, “Integrating organizational control into multi-agent learning,” in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2009, pp. 757–764.

[33] C. Zhang, V. Lesser, and S. Abdallah, “Self-organization for coordinating decentralized reinforcement learning,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1*, 2010, no. Aamas, pp. 739–746.

[34] X. Cheng, J. Shen, H. Liu, and G. Gu, “Multi-robot cooperation based on hierarchical reinforcement learning,” in *Computational Science–ICCS 2007*, 2007, pp. 90–97.

[35] C. Amato, G. Konidaris, and L. Kaelbling, “Planning with macro-actions in decentralized POMDPs,” in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 2014, pp. 1273–1280.

[36] P. Trigo and H. Coelho, “A hybrid approach to teamwork,” in *Proc. VI Encontro Nacional de Inteligncia Artificial (ENIA-07)*, 2007.

[37] D. Ye, M. Zhang, and D. Sutanto, “Self-organization in an agent network: A mechanism and a potential application,” *Decis. Support Syst.*, vol. 53, no. 3, pp. 406–417, Jun. 2012.

[38] Q. P. Lau, M. L. Lee, and W. Hsu, “Distributed Coordination Guidance in Multi-agent Reinforcement Learning,” *2011 IEEE 23rd Int. Conf. Tools with Artif. Intell.*, pp. 456–463,

Nov. 2011.

[39] X. Sun, L. E. Ray, J. D. Kralik, and D. Shi, “Socially augmented hierarchical reinforcement learning for reducing complexity in cooperative multi-agent systems,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, pp. 3244–3250.

[40] C. Amato, G. Konidaris, and G. Cruz, “Planning for Decentralized Control of Multiple Robots Under Uncertainty,” *arXiv Prepr. arXiv …*, pp. 1241–1248, 2014.

[41] R. D. McKelvey, A. M. McLennan, and T. L. Turocy, “Gambit: Software Tools for Game Theory, Version 0.2010.09.01.” 2011.

[42] J. C. Harsanyi and R. Selten, *A general theory of equilibrium selection in games.*, 1st ed. MIT Press Books, 1988.

[43] T. L. McKelvey, Richard D., McLennan, Andrew M., and Turocy, “Gambit: Software Tools for Game Theory.” 2014.

[44] “The Gambit Project.” [Online]. Available: https://www.google-melange.com/gsoc/org2/google/gsoc2012/gambit. [Accessed: 04-Apr-2016].

[45] M. Shor, “GameTheory.net.” [Online]. Available: http://www.gametheory.net/applets/solvers.html. [Accessed: 04-Jun-2016].

[46] D. K. Levine, “Economic and Game Theory Zero Sum Game Solver.” [Online]. Available: http://levine.sscnet.ucla.edu/games/zerosum.htm#Enter. [Accessed: 04-Apr-2016].

[47] “Calculating the Solution of a Matrix Game.” [Online]. Available: https://www.math.ucla.edu/~tom/gamesolve.html. [Accessed: 04-Apr-2016].

[48] R. Savani and B. von Stengel, “Game Theory Explorer: software for the applied game theorist,” *Comput. Manag. Sci.*, pp. 1–29, 2014.

Biographical Information

Danielle M. Clement received her Bachelor of Science from the University of Richmond in Computer Science, Mathematics and Physics in 2000. She received her Masters and Ph.D. in Computer Science Engineering from the University of Texas at Arlington in 2006 and 2016 respectively. Her research interests include team formation and dynamics, knowledge representation and visualization, and software processes and metrics.