9-2011

Scaling Multi-Agent Learning in Complex Environments

Chongjie Zhang
University of Massachusetts Amherst, chongjie@cs.umass.edu

SCALING MULTI-AGENT LEARNING IN COMPLEX ENVIRONMENTS

A Dissertation Presented

by

CHONGJIE ZHANG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2011

Department of Computer Science

SCALING MULTI-AGENT LEARNING IN COMPLEX ENVIRONMENTS

A Dissertation Presented

by

CHONGJIE ZHANG

Approved as to style and content by:

Victor Lesser, Chair

Andrew Barto, Member

Shlomo Zilberstein, Member

Jonathan Machta, Member

Andrew Barto, Department Chair
Department of Computer Science

ACKNOWLEDGMENTS

I am deeply grateful to my advisor Victor Lesser, who has given his time and talent in ways that simply cannot be adequately acknowledged. Victor has provided constant, immeasurable intellectual challenge and inspiration. His energy and enthusiasm are renowned and I benefited greatly from them time and again. Victor has served all of the roles of teacher, guide, encourager, and friend. He has not only cared about my research development, but also my personal life and my family, especially at difficult times. I am very happy to be one of his students, and I am thankful for the opportunity to learn from Victor.

I also especially want to thank my other thesis members Professors Andrew Barto, Shlomo Zilberstein, and Jon Machta. They have served to greatly improve the content and presentation of this thesis. I am grateful to Andrew Barto for his insightful and rigorous comments both on my work and on this thesis and for his tremendous support. I am thankful to Shlomo Zilberstein for his insightful questions on this thesis and for his motivating discussions. I would also like to thank Jon Machta for probing and stimulating questions during my defense.

I am indebted to our Multi-Agent Systems (MAS) Lab. Dan Corkill has provided insightful comments for every paper I submitted. I enjoyed his countless motivating discussions about research and programming and thank him for sharing his insight, wisdom, and practical advice. I also would like to thank former MAS members, Sherief Abdallah, Shelley Zhang, and Haizheng Zhang, with whom I have been collaborating over the years. I want to thank my colleagues in MAS lab: Bo An, Hala Mostafa, Yoonheui Kim, and Huzaifa Zafar, and in Resource-Bounded Reasoning Lab: Akshat Kumar, Alan Carlin, William Yeoh and Chris Amato, for many interesting and useful discussions. I thank our lab manager Michele Robert for assisting in a variety of administrative paperworks.

I owe the Department of Computer Science at the University of Massachusetts Amherst (UMass), a great deal of gratitude for their extended support. UMass was a good fit in balancing my life as a graduate student, and a family guy. I particularly thank those professors who have taught or worked with me over these years that have been the most rewarding educational experience of my life: Andrew Barto, Shlomo Zilberstein, Sridhar Mahadevan, Neil Immerman, Prashant Shenoy, David Jensen, and Andrew McCallum.

Finally, I would like to thank my family, whose encouragement and timely distractions kept me sane, productive, and allowed me to thoroughly enjoy my time as a graduate student. I thank my parents, my in-laws, my brother, and my sister for their endless support and encouragement for pursuing what I am capable of. I also thank my wonderful daughter, Sophia, who has been rooting for me like nobody else. I thank her for her tolerance and her smiles. Most of all, my wife, Xiaoxi Xu, is an essential partner to whatever success I am able to claim. She inspires me and envisions things for me that I could never imagine alone. She has encouraged, cajoled, and hounded me, as necessary, to bring this work to fruition. Thanks for seeing this through with me.

ABSTRACT

SCALING MULTI-AGENT LEARNING IN COMPLEX ENVIRONMENTS

SEPTEMBER 2011

CHONGJIE ZHANG

B.Sc., UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

M.Sc., LOUISIANA STATE UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor Lesser

Cooperative multi-agent systems (MAS) are finding applications in a wide variety of domains, including sensor networks, robotics, distributed control, collaborative decision support systems, and data mining. A cooperative MAS consists of a group of autonomous agents that interact with one another in order to optimize a global performance measure. A central challenge in cooperative MAS research is to design distributed coordination policies. Designing optimal distributed coordination policies offline is usually not feasible for large-scale complex multi-agent systems, where 10s to 1000s of agents are involved, there is limited communication bandwidth and communication delay between agents, agents have only limited partial views of the whole system, etc. This infeasibility is either due to a prohibitive cost to build an accurate decision model, or a dynamically evolving environment, or the intractable computation complexity.

This thesis develops a multi-agent reinforcement learning paradigm to allow agents to effectively learn and adapt coordination policies in complex cooperative domains without

explicitly building the complete decision models. With multi-agent reinforcement learning (MARL), agents explore the environment through trial and error, adapt their behaviors to the dynamics of the uncertain and evolving environment, and improve their performance through experiences. To achieve the scalability of MARL and ensure the global performance, the MARL paradigm developed in this thesis restricts the learning of each agent to using information locally observed or received from local interactions with a limited number of agents (i.e., neighbors) in the system and exploits non-local interaction information to coordinate the learning processes of agents. This thesis develops new MARL algorithms for agents to learn effectively with limited observations in multi-agent settings and introduces a low-overhead supervisory control framework to collect and integrate non-local information into the learning process of agents to coordinate their learning. More specifically, the contributions of already completed aspects of this thesis are as follows:

Multi-Agent Learning with Policy Prediction: This thesis introduces the concept of policy prediction and augments the basic gradient-based learning algorithm to achieve two properties: best-response learning and convergence. The convergence property of multi-agent learning with policy prediction is proven for a class of static games under the assumption of full observability.

MARL Algorithm with Limited Observability: This thesis develops PGA-APP, a practical multi-agent learning algorithm that extends Q-learning to learn stochastic policies. PGA-APP combines the policy gradient technique with the idea of policy prediction. It allows an agent to learn effectively with limited observability in complex domains in presence of other learning agents. The empirical results demonstrate that PGA-APP outperforms state-of-the-art MARL techniques in both benchmark games.

MARL Application in Cloud Computing: This thesis illustrates how MARL can be applied to optimizing online distributed resource allocation in cloud computing. Empirical results show that the MARL approach performs reasonably well, compared

to an optimal solution, and better than a centralized myopic allocation approach in some cases.

A General Paradigm for Coordinating MARL: This thesis presents a multi-level supervisory control framework to coordinate and guide the agents' learning process. This framework exploits non-local information and introduces a more global view to coordinate the learning process of individual agents without incurring significant overhead and exploding their policy space. Empirical results demonstrate that this coordination significantly improves the speed, quality and likelihood of MARL convergence in large-scale, complex cooperative multi-agent systems.

An Agent Interaction Model: This thesis proposes a new general agent interaction model. This interaction model formalizes a type of interactions among agents, called *joint-even-driven* interactions, and define a measure for capturing the strength of such interactions. Formal analysis reveals the relationship between interactions between agents and the performance of individual agents and the whole system.

Self-Organization for Nearly-Decomposable Hierarchy: This thesis develops a distributed self-organization approach, based on the agent interaction model, that dynamically form a nearly decomposable hierarchy for large-scale multi-agent systems. This self-organization approach is integrated into supervisory control framework to automatically evolving supervisory organizations to better coordinating MARL during the learning process. Empirically results show that dynamically evolving supervisory organizations can perform better than static ones.

Automating Coordination for Multi-Agent Learning: We tailor our supervision framework for coordinating MARL in ND-POMDPs. By exploiting structured interaction in ND-POMDPs, this tailored approach distributes the learning of the global joint policy among supervisors and employs DCOP techniques to automatically coordinate distributed learning to ensure the global learning performance. We prove that

this approach can learn a globally optimal policy for ND-POMDPs with a property called *groupwise observability*.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

A cooperative multi-agent system (MAS) is composed of a set of autonomous agents that interact with one another in a shared environment in order to reach a shared goal or optimize a global performance measure. Each agent perceives the state of the environment through its sensors, makes decisions, and acts upon the environment with its actuators. Cooperative multi-agent systems are finding applications in a wide variety of practical domains, including sensor networks, robotics, distributed control, collaborative decision support systems, supply chains, and data mining. They arguably offer the most natural way of viewing, characterizing, and realizing many distributed, dynamic, and open cooperative intelligent systems. For example, in sensor networks or robotic teams, because of limited communication bandwidth, the control authority is naturally distributed among sensors or robots, which work together to achieve some common goal (e.g., tracking vehicles or weather phenomena). In electricity grids, electricity distribution management is decentralized among power stations, which coordinate their power control configurations in order to satisfy variable demands from all customers and minimize losses.

A central challenge in cooperative MAS research is to design distributed decision policies for agents to coordinate their actions in order to efficiently achieve their common goal. A common offline approach is to build a model (e.g., decentralized Markov decision process) for distributed decision problems in a cooperative MAS and then compute coordination policies for agents from the model. However, this approach is usually infeasible for large-scale complex MAS applications, which involve tens to thousands of agents with limited communication bandwidth and partial views of the whole system. Firstly, it is very

expensive, time-costly, or even not possible to obtain an accurate model of practical MAS applications. This is especially true for applications operating in open environments where the environmental characteristics are not known a priori and may evolve over time. Secondly, even when we have such models, the computation for optimal policies for agents is usually intractable.

Multi-agent reinforcement learning (MARL) potentially provides an attractive approach for agents to developing effective coordination policies without explicitly building a complete decision model. MARL allows agents to explore environment through trial and error, adapt their behaviors to the dynamics of the uncertain and evolving environment, and gradually improve their performance through experiences. MARL has gained a great deal of interest over the recent years, but its open research challenges are still in flux. One key challenge of MARL is its non-stationary environment where agents are concurrently learning and adapting to one another. This non-stationarity violates a fundamental assumption for the convergence guarantee of most existing learning techniques. Therefore, new algorithms may be required that are tailored MARL and deal with non-stationarity. The convergence guarantee of existing MARL algorithms [55, 39, 56, 29, 92, 23, 133, 22, 3, 124] is limited to a limited MAS settings. In addition, these algorithms mainly focus on whether they converge or not to an equilibrium, but not on which equilibrium they converge to, which is especially important for cooperative MAS. Another key challenge of MARL is its scalability to realistic problems, which, already problematic in single-agent reinforcement learning, is an even greater cause for concern in large-scale multi-agent settings. Many state-of-the-art techniques [59, 17, 31, 46, 67, 102] to speeding up or scaling up MARL are either restricted to specific domains or not scalable in large agent networks. In summary, state-of-the-art techniques for MARL are still inadequate to address the MARL challenges in large-agent complex cooperative systems.

Although realistic MAS application systems are very large and complex, they usually possess some structures of interactions among agents. One common interaction property,

called *interaction locality*, exists in most large systems, where each agent interacts with only a limited number of neighboring agents. As argued by Herbert Simon [90] in "Architecture of Complexity", many complex systems also have a nearly decomposable, hierarchical structure. A nearly decomposable systems contains sub-systems, where interactions within the subsystems are strong, while interactions between themselves are relative weak but not negligible. Human decision makers often exploit such a type of structure and form organizations (e.g., companies) to solve large-scale problems (e.g., managing manufacturing factories).

This thesis develops a new learning paradigm that *exploits interaction locality and non-local supervisory control in a coherent way to scale up MARL in complex domains*. This paradigm employs MARL algorithms for agents to learn local coordination policies only based on information from their local interactions. To improve the overall learning performance, this paradigm uses an emergent supervisory organization with low overhead that exploits non-local information to dynamically coordinate and shape the learning processes of individual agents while still leaving agents to react autonomously to local feedbacks. This thesis addresses two aspects of this learning paradigm: designing effective MARL algorithms for agents to learn with limited observations in a non-stationary environment, and developing a multi-level supervisory control framework to collect and integrate non-local information into the learning process of agents. We empirically demonstrate that our paradigm yields effective performance in diverse large-scale problem domains, including distributed task allocation, network routing, and sensor networks.

1.1 Motivating Examples

The focus of this thesis is developing a new paradigm for applying MARL to large-scale, autonomous, cooperative multi-agent systems. To motivate the research from the practical perspective, this section describes examples of application domains where the approaches developed in this thesis can be applied.

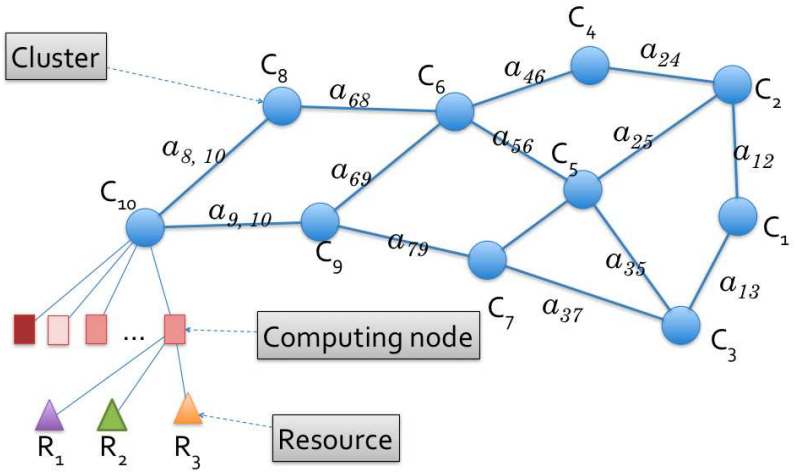


Figure 1.1. A network of shared clusters

1.1.1 Cloud Computing

As "Software as a service" becomes a popular business model, it is becoming increasingly difficult to build large cloud computing infrastructures that can host effectively the wide spread use of such services. *Shared clusters* built using commodity PCs or workstations offer a cost-effective solution for constructing such infrastructures. Unlike a dedicated cluster, where each computing node is dedicated to a single application, a shared cluster can run the number of applications significantly larger than the number of nodes, necessitating resource sharing among applications. Resource management approaches developed for shared clusters [8, 7, 113] are centralized, which limits the cluster scale.

To build larger shared computing infrastructures, one common model is to organize a set of shared clusters into a network and enable resource sharing across shared clusters, as shown in Figure 1.1. Each cluster C_k is regarded as an agent. Each agent has a set of computing nodes, each of which provides a set of resources. The label a_{ij} on the edge between agent C_i and C_j indicates communication delay. The resource allocation decision is now distributed to each agent. Each agent still uses a cluster-wide technique for managing its local resources. Each agent receives tasks from either the external environment or a neighbor. At each time step, an agent makes decisions on what tasks are allocated locally

and to which neighbors the tasks not allocated locally should be forwarded. To reduce the communication overhead, the number of tasks an agent can transfer at each time step is limited. To allocate a task, an agent should have available resources to satisfy its resource requirements. When a task is allocated locally, the agent gains utility at each time step, which is specified by the task utility rate. If an allocated task is finished, all resources it occupies will be freed and available for future tasks. The main goal of the system is to derive decision policies for each agent that maximize the average utility rate of the whole system. The effectiveness of one agent's policy is dependent on the policies of other agents, thus the need for coordinated policies of agents.

As will be discussed in Chapter 4, a MARL algorithm was used for each agent to learn and adapt its decision policies in such an environment. To simplify the learning, each agent's decisions were decomposed into two connected learning problems: local allocation problem (deciding what tasks to be allocated locally) and task routing problem (deciding where to forwarded a task). Because of limited communication bandwidth, it was not feasible to compute real-time global performance measure for agents. Therefore, the learning of each agent is based on its local observations and reward signals. To avoid poor initial policies during learning, heuristic strategies were developed to speed up the learning in both problems. Experimental results showed that the MARL approach worked effectively and even outperformed the centralized approach in some cases. However, MARL did not scale well and converged slowly with even 16 agents, which motivated the research of this thesis to scale up MARL.

1.1.2 Sensor Networks

Advances in processor, memory and radio technology have enabled cheap sensors capable of sensing, communication and processing. Due spatial diversity and fast response, a network of small, inexpensive, low-performance sensors may be able to outperform a system using a single, very expensive, high-performance sensor. As a result, networks

of distributed sensors are now rapidly emerging as a feasible solution to a wide range of data gathering applications, such as tracking weather phenomena [48] and vehicle movement [36].

To optimize sensing performance, networked sensors needs to be coordinated among themselves. Due to limited communication bandwidth and battery power, the coordination control of sensor network is decentralized, which leads to large-scale distributed decision-making problems. Each sensor not only decides where to sense, but also decisions over when to communicate information to other sensors, and when to sense the environment, thus maximizing the amount and quality of information gathered, while bounding power consumption.

Sensor networks are basically cooperative multi-agent systems. Due to uncertainty in environments and partial observability of sensors, decision-making problems in sensor networks can be modeled as Decentralized Partially Observable Markov Decision Process (DEC-POMDP) or its restricted versions (e.g., Networked Distributed POMDP (ND-POMDP)) [114]. Offline techniques [114, 61, 49] has been developed for such problems. However, these techniques require accurate models, which are usually costly to obtain in practice. As will be discussed in Chapter 8, our coordinated multi-agent learning approach can be applied to such problems, which does not need to build an environment model and can scale to larger networks.

1.1.3 Other Domains

Cooperative multi-agent learning can find applications in many real-world problems in addition to our motivating example. We present some examples of complex practical problems where our MARL paradigm can be applied.

Queuing networks Queueing problems are a special type of stochastic dynamic system, where an agent who manages a set of queues of jobs must decide which one to serve at every time step. These problems have been widely studied in the literature, as they

provide abstractions of many practical problems in industry. Queueing networks [18] are an extension of this model to problems involving many agents (servers) simultaneously.The network defines a process where jobs that are served in one queue are then assigned to another one. The cloud computing example is one type of queuing networks.

Network Management and Routing There are many possible applications of MARL algorithms in networking tasks (e.g., packet routing [24]). An interesting potential application is the routing of queries in peer-to-peer systems [130]. We can consider each node as an agent that can decide to fulfill a query, or forward it to one of the neighboring nodes. The state of this system is specified by the information (e.g., documents) stored in each node and the query. A node cannot observe (or even store) the state of every node in the network, and should not flood the entire system with queries. Using our learning approach, we could tackle such problems effectively, requiring only limited observability and limited communication between the nodes.

Electricity Distribution Management Here the problem is to maintain an optimal power grid configuration that all customers supplied and minimizes losses; while at the same time dealing with possible damage to the network, variable demand from customers, scheduled maintenance operations, and equipment failures and upgrades [274]. Schneider et al. [230] present a reinforcement learning approach to managing a power grid.

Computing Games In recent years, there has been a significant increase in interest in the applications of AI techniques to computer games. There are many games that require distributed decision-making for coordination. One of such games is called Freecraft, a strategic war game and an open-source version of the popular Warcraft game. The objective of this game is to coordinate the actions of a set of units (agents) with different skills in order to defeat an enemy force.

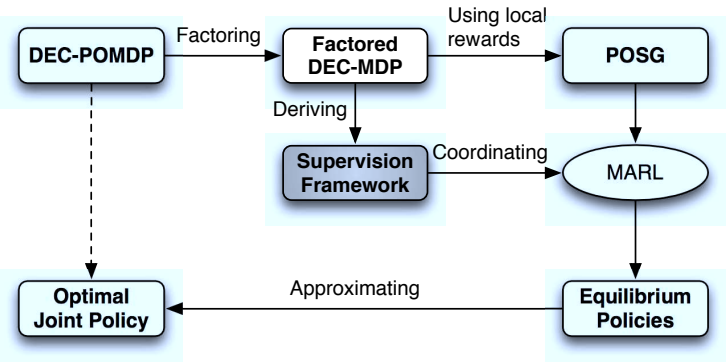


Figure 1.2. A encompassing view of our MARL paradigm

1.2 Approach

As with application examples described in previous section, manu realistic multi-agent systems are large, dynamic, and complex: they involve 10s to 1000s of agents, there is limited communication bandwidth and communication delay between agents, and agents have only limited partial views of the whole system, etc. This thesis seeks to answer the following key question: *can a network of agents effectively learn to perform collectively in such complex cooperative domains to optimize the global performance*?

The decision-making optimization problem that arises in cooperative multi-agent systems can be generally formulated as a decentralized partially observable Markov decision process (DEC-POMDP) [15]. In a DEC-POMDP, all agents share a global reward function. It seems natural to use this global reward signal for agents to learn in cooperative systems. However, due to limited communication bandwidth in most realistic systems, it is often infeasible to calculate the global reward signal in a real-time manner for the learning. In addition, the global reward signal is usually not sufficiently tailored to the behavior of an individual agent, resulting in learning very slowly [120]. As shown in Figure 1.2, we use a factored DEC-MDP to approximate a general DEC-POMDP by designing local reward signals for agents, which are efficiently computable and sufficiently tailored to their behaviors. As will be defined in Chapter 7, a factored DEC-MDP further assumes the global state

is factored by agents' local states and each agent fully observes its local state. By using local reward signals, the learning essentially treats a DEC-MDP as a partially observable stochastic game (POSG) [34]. POSG is a generalization of DEC-POMDP, where every agent can have its own reward function. Multi-agent learning algorithm is employed to learn equilibrium policies in the POSG, which can *be stochastic*. The learned equilibrium policies are used as an approximate solution for the original DEC-POMDP problem. As illustrated in application examples described in previous section, the property of *interaction locality* commonly exists in many practical large-scale collaborative multi-agent systems, where each agent directly interacts with only a limited number of neighboring agents. By exploiting this property, our MARL paradigm restricts the learning of each agent to using local rewards and observations from local interactions so that its policy space will not explode exponentially as the number of agents increases.

As shown in Figure 1.2, in our MARL paradigm, we also develop a supervision framework, which derived from the factored DEC-MDP model. This is because exploiting interaction locality alone usually can not address all challenges faced by MARL in complex cooperative MAS applications. For example, due to non-stationary learning environments, theoretical convergence results of existing learning algorithms do not hold for general POSGs. In addition, as a POSG usually have multiple equilibria, MARL may converge to an equilibrium that yields very poor global performance. Further more, even with a limited neighborhood of agents, the policy space of each learning agent still remains large because of the need to represent characteristics of neighbors.

Our supervision framework has the potential to address these challenges of scaling MARL. It exploits non-local information and low-overhead, periodic multi-level organizational control to coordinate and guide the agents' learning process. This supervision framework introduces a more global view into the learning process of individual agents without incurring significant overhead and exploding their policy spaces; it coordinates the learning behavior of tightly coupled agents by constraining their learning processes while still leav-

Figure 1.3. A scalable, approximate learning approach for cooperative MAS

ing agents to react autonomously to local reward signals. This coordination results in both speeding up and increasing the likelihood of convergence by reducing the occurrence of oscillatory behavior among agents learning in a non-stationary environment and focusing agents' exploration. Additionally, it also results in improved overall solution quality due to coordination directives that are based on a more global view of current learning.

In summary, as illustrated in Figure 1.3, our paradigm first uses MARL for agents to learn local coordination policies, which exploits interaction locality and local or neighboring information to restrict the size of policy search spaces of individual learning agents, and then uses a low-overhead organizational control framework, which exploits nearly decomposable structure and non-local information, for coordinating learning processes of individual agents to ensure its global learning performance. This thesis will present techniques for both aspects of our paradigm: effective MARL algorithms for limited observability and the general supervision framework for coordinating MARL.

1.2.1 Multi-Agent Learning with Limited Observability

By learning, we mean the process of an agent adapting its behavior through experience to improve its ability to achieve a goal or maximize long-term reward. Learning occurs

through the agent's interaction with the environment: observing percepts and gaining rewards from the world, and taking actions to affect it. In a MAS, the learning of an agent is complicated by the presence of other agents concurrently acting, adapting and affecting the environment, which is referred as *multi-agent learning* (MAL) [1]. As agents interact and concurrently learn their policies, the environment becomes non-stationary from the perspective of each individual agent. The convergence guarantee for single-agent reinforcement learning does not hold for multi-agent settings.

In this thesis, one of our goals is to develop effective multi-agent learning algorithms for agents with limited information about other agents and the environment. We first empirically investigate multi-agent learning in complex cooperative MAS applications. As will be discussed in Chapter 4, we design a gradient-based MARL algorithm that extends Q-learning with the idea from Generalized Infinitesimal Gradient Ascent (GIGA) algorithm [133]. This algorithm only requires each agent observing its immediate reward for selecting an action. We apply this algorithm to optimizing online distributed task allocation in Cloud Computing. Empirical results demonstrate an impressive performance of this algorithm in convergence and solution quality. However, as with GIGA, this algorithm does not converge in competitive scenarios, which can occur in cooperative MAS when we design local reward signals (instead of using the single global reward) for agents' learning.

To consider both cooperative and non-cooperative scenarios, we introduce the concept of policy prediction and augment the basic GIGA algorithm. In this thesis, we study our multi-agent learning algorithms in the framework of stochastic games. Stochastic games are POSGs where all agents can observe the state of the environment. Although fully observability may not be realistic for many practical MAS application, MAL algorithms developed for stochastic games usually also perform well in POSGs.

[1] In this thesis, we will use MAL and MARL interchangeably

Stochastic games were first studied extensively in the field of game theory, but can be viewed as a generalization of Markov decision processes (MDPs). MDPs have served as the foundation of much of the research in single-agent control learning. Stochastic games subsume both MDPs as well as the more well-known game theoretic model of static games (or matrix games). Like static games, stochastic games do not always have a well-defined notion of optimal behavior for a particular agent, since its performance may depend on behaviors of other agents. The most common solution concept in these games is Nash equilibria, intuitively defined as a particular behavior for all the agents where each agent is acting optimally with respect to the other agents's behavior. All stochastic games have at least one Nash equilibria. However, Nash equilibria are only sensible if all the agents are fully rational, optimal, and unlimited (having fully observations and complete knowledge about other agents and the environment). Although, in this thesis, we do not make Nash equilibria the explicit goal of multi-agent learning, we will use it as a tool to understand the dynamics of multi-agent learning algorithms.

We first examine the convergence property of our MAL algorithm in stochastic games when agents have unlimited observations of other agents and the environment. We then consider how limited observability of the learning agent affects the multi-agent learning scenario. We extend the techniques developed in the unlimited setting to address the more challenging problem of learning with only observation of the reward signal of choosing a given action. Our goal is to develop MAL techniques that are theoretically justifiable in analyzable and unlimited settings but still perform effectively in practical and challenging problems.

As we design MAL algorithms, we seek to avoid making assumptions about the goals or rewards of the other agents, and do not expect a priori that their behaviors are cooperative or adversarial. This generality, however, may prevent this aspect of our thesis work from being the most appropriate approach for some multi-agent environments where stricter assumptions can be potentially exploited, such as cooperative multi-agent systems. The next

section will focus on addressing challenges of MARL in complex cooperative multi-agent systems.

1.2.2 Coordinating Multi-Agent Learning

Developing effective MARL algorithms are essential to address MARL challenges. Existing MARL algorithms, including those developed in this thesis, can potentially scale up in large MASs by exploiting interaction locality so that each agent learns based on its local observations and local reward signals. However, there are still challenges faced by MARL in large complex cooperative MAS applications. Firstly, the theoretical convergence guarantee of existing MAL algorithms is still limited for special classes of stochastic games (e.g., repeated static games), and does not hold for general POSGs. Secondly, due to interaction locality and communication delay, agents have limited and even outdated views of the system. Thirdly, the "tragedy of the commons" problem often exists for MARL using local reward signals in cooperative MASs, that is, greedy policies at agents can harm the global performance. Finally, even with a limited neighborhood of agents, the policy space still remains large because of the need to represent characteristics of neighboring agents. As a result, MARL may converge slowly, converge to inferior policies, or even diverge in realistic settings.

In this thesis, we develop a supervision framework for coordinating MARL to tackle these challenges (see Figure 1.4 and 1.5). This framework exploits non-local information and uses low-overhead multi-level organizational control to coordinate and guide learning processes of agents. The supervision framework defines a multi-level organizational structure and a communication protocol for exchanging information between lower-level agents (or subordinates) and higher-level supervising agents (or supervisors) within an organization. As shown in Figure 1.4, subordinates periodically (e.g., every 500 learning cycles) report their abstract states and rewards to their supervisors, who then generate abstract states of their own clusters and exchange them with neighbors. Based on abstracted states

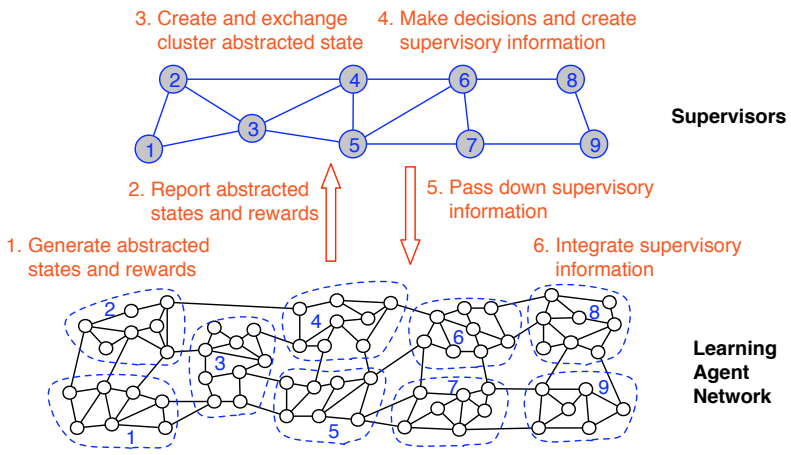


Figure 1.4. A supervision process of the organization-based control framework

of their subordinates and neighboring clusters, supervisors create and pass down supervisory information. Subordinates then integrate supervisory information into their learning processes, which will be conducted in a coordinated way. This framework is general and can be used with most existing MARL algorithms, including those developed in this thesis. This framework exploits a hierarchy of control and data abstractions, which is conceptually different from existing hierarchical multi-agent learning algorithms that use a hierarchy of task abstractions [59].

This supervision framework has the potential to address MARL challenges in complex cooperative MASs. To deal with non-stationarity, this framework exploits non-local information and guides and coordinates agents' exploration of their state-action spaces to reduce occurrence of oscillatory behaviors and improve the likelihood of convergence. It can also provide non-information for learning agents to expand their local views and speed up information propagation in the system. Through supervisory information generated from a more global view, supervisors can force or steer learning agents to make joint movement of their policy updates to deal with "tragedy of the commons" problems and allow them to converge to a better equilibrium. To speed up learning processes of agents with large

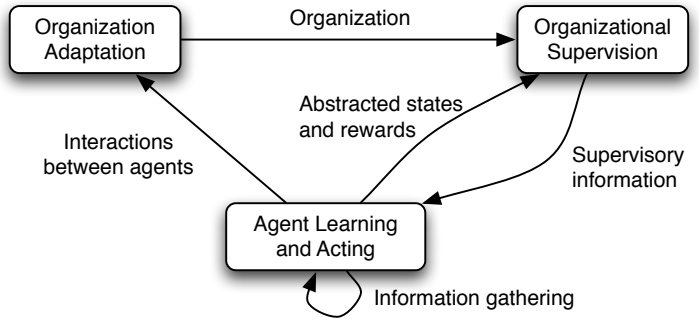


Figure 1.5. Dynamic supervision framework with self-organization

policy search spaces, this framework uses supervisory information to have learning agents focus their exploration in certain subset of their policy space.

To facilitate this supervision framework to be applied to practical MARL applications, in this thesis, we also attempt to address two important problems of this framework: finding supervisory organizations for coordinating agents' learning processes and automating supervision process to generate supervisory information with little or no domain knowledge.

Self-Organization Through experiments, we observe that different supervisory organizations yield different performance for coordinating MARL. Interesting questions arise from this observation: can supervisory organizations automatically form while agents are concurrently learning their decision policies? do such dynamically evolving organizations perform better than static supervisory organizations? The key problem of forming supervisory organizations is to decide which agents need to be clustered together so that their exploration strategies can be coordinated. Inspired by the concept of *nearly decomposable systems* [89] (where interactions between subsystems are generally weaker than interactions within subsystems), we develop a self-organization approach to dynamically forming a nearly decomposable hierarchical structure. Such a hierarchical structure can potentially reduce coordination complexity and improve coordination quality. In our approach, we first develop an agent interaction model based on factored DEC-MDP. The interaction model characterizes a type of agent interactions and defines a measure for capturing the strength

of interactions among agents, given their current state of learning. Based on the interaction model, we then design a negotiation-based self-organization algorithm that incrementally group agents together that strongly interact with each other and adapts supervisory organizations for coordinating MARL during the learning process. Experimental results show that our dynamically evolving organizations outperform predefined organizations for coordinating MARL.

Automating Supervision Our supervision framework provides a way of integrating domain knowledge to improve the MARL performance through dynamically generating supervisory information based on agents' learning status. To broaden the applicability of this supervision framework, it is highly desirable to automate the process of generating supervisory information for supervisors. In this thesis, our attempt in this research direction focuses on a class of cooperative multi-agent decision making problems, which can be modeled by Networked Distributed POMDPs (ND-POMDPs) (a restricted version of DEC-POMDP). We tailor our supervision framework for coordinating MARL in ND-POMDP problems: making supervisors learn policies for their own subordinates and employing distributed constraint optimization (DCOP) techniques to automatically coordinate supervisors' learning without employing domain knowledge or heuristics. By using a message-passing DCOP algorithm, this approach can be implemented in a distributed way as an anytime algorithm that trades off solution quality and cost of communication and computation. We formally prove its convergence and optimality were for a restricted class of ND-POMDPs and empirically demonstrate its effectiveness in sensor networks for tracking mobile targets.

1.3 Main Contributions

This thesis makes a number of important contributions to the state of the art in the area of multi-agent learning by looking at large-scale complex multi-agent decision-making

problems from both theoretical and heuristic perspectives. The main contributions can be summarized as follows:

MARL Application in Cloud Computing: Resource allocation in computing clusters are traditionally centralized, which limits the cluster scale. Effective resource allocation in a network of computing clusters may enable building larger computing infrastructures for cloud computing. We design a simple gradient-based MARL algorithm that extends Q-learning to learning stochastic policies. We illustrate how this algorithm can be applied to optimizing this online distributed resource allocation problem. The learning is distributed to each cluster, using local information only and without access to the global system reward. We empirically show that the MARL approach performs reasonably well, compared to an optimal solution, and better than a centralized myopic allocation approach in some cases. This part of the work was published in IJCAI 2009 [129].

Multi-Agent Learning with Policy Prediction: Best response and convergence are two properties desirable for a MARL algorithm. We introduce the concept of policy prediction and augment the basic gradient-based learning algorithm to achieve these two properties. We demonstrate that multi-agent learning with policy prediction is theoretically grounded in a class of general-sum stochastic games under the assumption of full observability. The first stages of this work was published in AAAI 2010 [124].

MARL Algorithms with Limited Observability: We present a policy gradient learning technique extending Q-learning to learn stochastic policies in multi-agent settings. This policy gradient technique is augmented with the idea of policy prediction. The resulting new MARL algorithm is intended to address the question of how an agent effectively learns with limited observability in complex domains with other learning agents. The empirical results demonstrate that our new MARL algorithm out-

performs state-of-the-art MARL techniques in both benchmark games and complex problems. This algorithm was published in AAAI 2010 [124].

A General Paradigm for Coordinating MARL: We introduce a new paradigm that builds upon conventional MARL techniques for scaling MARL to large agent networks. This paradigm is based on a multi-level supervisory control framework to coordinate and guide the agents' learning process. This framework exploits non-local information and introduces a more global view into the learning process of individual agents without incurring significant overhead and exploding their policy spaces; it coordinates the learning behavior of tightly coupled agents by constraining their learning processes while still leaving agents to react autonomously to local reward signals. This coordination during learning results in both speeding up and increasing the likelihood of convergence by reducing the occurrence of oscillatory behavior among agents learning in a non-stationary environment and focusing agents' exploration. Additionally, it also results in improved overall quality of learned coordination policies due to supervisory coordination directives that are based on a more global view of current learning. This was published in AAMAS 2009 [123].

An Agent Interaction Model: We propose a new general agent interaction model based on a decentralized Markov decision process (DEC-MDP) model (which generalizes distributed decision-making problems in cooperative MAS). Our interaction model formalizes a type of interactions among agents, called *joint-even-driven* interactions, and define a measure for capturing the strength of such interactions. We formally analyze how interactions between agents affect the performance of individual agents and the whole system. This interaction model can be used to decompose decision-making problems in large-scale multi-agent systems and simplify the complexity of coordinating agents' behaviors. This was published in AAMAS 2010 [128].

Self-Organization for Nearly-Decomposable Hierarchy: We develop a distributed self-organization approach, based on our agent interaction model, that dynamically form a nearly decomposable hierarchy for large-scale multi-agent systems. We extends our supervisory control framework to integrate this self-organization approach to automatically evolving supervisory organizations to better coordinating MARL during the learning process. Empirically results show that dynamically evolving supervisory organizations can perform better than static ones. This was published in AAMAS 2010 [128].

Automating Coordination for Multi-Agent Learning: We tailor our supervision framework for coordinating MARL in ND-POMDPs. By exploiting structured interaction in ND-POMDPs, this tailored approach distributes the learning of the global joint policy among supervisors and employs DCOP techniques to automatically coordinate distributed learning to ensure the global learning performance. We prove that this approach can learn a globally optimal policy for ND-POMDPs with a property called *groupwise observability*. Experimental results show that, with communication during learning and execution, our approach significantly outperforms the nearly-optimal non-communication policies computed offline. This work was published in AAAI 2011 [125].

1.4 Guide to the Thesis

The rest of this thesis is structured into four parts. The first part describes background knowledge, which contains Chapter 2 and 3. Chapter 2 introduces formal frameworks for studying multi-agent learning that will provide the background necessary to understand the detail research we will present in later chapters. Chapter 3 discusses related research on multi-agent learning in cooperative systems. The second part presents our work in multi-agent reinforcement learning algorithms, which contains Chapter 4 and 5. Chapter 4 provides a simple gradient-based MARL algorithm and applies it to optimiz-

ing distributed task allocation in Cloud Computing. Chapter 5 presents our multi-agent learning algorithms which are both formally analyzed and empirically evaluated. The third part discusses our work in coordinating multi-agent learning, which contains Chapter 6, 7, and 8. Chapter 6 presents a supervision framework exploits low-overhead, periodic, non-local multi-level organizational control to coordinate and guide the agents' learning process to improve MARL performance in cooperative systems. Chapter 7 describes a self-organization approach built on a new agent interaction model to dynamically evolving supervisory organizations to better coordinate agents' learning processes. Chapter8 show that distributed constraint optimization techniques can be used to automate coordinating MARL in ND-POMDPs. The last part is Chapter 9 summarizes our work in this thesis and discusses future research directions.

Part I

BACKGROUND AND RELATED WORK

CHAPTER 2

MULTI-AGENT LEARNING FRAMEWORKS

A framework generalizes the structure of a class of problems with some assumptions and concepts. Multi-agent learning frameworks introduced in this chapter models the distributed decision making problems in multi-agent systems. They provides a formal foundation for generating, analyzing, and evaluating new multi-agent learning algorithms. In addition, using such frameworks, we can develop general techniques for improving multi-agent learning in a large class of cooperative systems.

In this thesis, we are studying multi-agent learning for solving decision-making problems in large-scale cooperative multi-agent systems, where agents work together to optimize the system performance. In particular, we focus on decentralized systems, where an agent has only a partial view of the system, that is, an agent does not have full observability of the state of all other agents in the system. The model of *decentralized partially observable Markov decision processes* (DEC-POMDP) generalizes such distributed problems. In a DEC-POMDP, all agents share the same reward function, which is called a *global reward function*. However, in many large-scale decentralized systems (e.g., network routing or distributed task allocation in our motivating example), learning agents do not have access to the global reward signals, because they can not be computed in real-time. Even when they are available for some systems, they are usually not specifically tailored to individual agents' performance and are not good feedbacks for agents' learning. Therefore, we need to design local reward signals, which are more specifically tailored for individual agents' behaviors and more easily computable. As shown in Figure 1.2, with local reward signals, learning in a DEC-POMDP is converted to learning in a more general framework,

partially observable stochastic game (POSG), where each agent can have its own reward function. The equilibrium solution of POSG is used as an approximate solution for DEC-POMDP. Therefore, we will study multi-agent learning in the POSG framework. A POSG can have multiple equilibria solutions, some of which may yields very bad global performance. The DEC-POMDP framework is useful for investigating general techniques for multi-agent learning to converge faster and a better equilibrium solution in POSGs that are converted from DEC-POMDPs by using local reward signals.

As with single-agent reinforcement learning studied in the MDP framework, in this thesis, we will focus the framework of *stochastic games*, which are POSGs where all agents fully observe the state of the environment. Multi-agent learning algorithms developed for stochastic games usually also perform well in POSGs. Stochastic games model multi-agent, multi-state sequential decision-making problems in both cooperative and non-cooperative systems. Although stochastic games were first introduced in the field of game theory, they are now being an increasingly popular framework to study multi-agent reinforcement learning (MARL). Many current MARL algorithms with theoretical convergence results, including our algorithm in Chapter 5, are initially designed and formally analyzed in one special cases of stochastic games, called *static games*. Static games consist of a set of interacting agents, each of which has a single state.

In this chapter, we will review these frameworks and some key concepts and results that we make use of later in this thesis. We begin in Section 2.1 with a very brief overview of the general model of an agent and agent learning, which underlies all of the frameworks we discuss. Although this thesis deals with multi-agent settings, it is still useful to understand the MDP model and single-agent reinforcement learning. This is because DEC-POMDPs can be viewed as a generalization of Markov decision processes (MDP) to the multi-agent cases and most practical MARL algorithms extends single-agent reinforcement learning (e.g., Q-learning). In Section 2.2, we describe the MDP model, the relevant solution concepts, and some standard learning algorithms. We then present the DEC-POMDP framework and dis-

cuss multi-agent learning in this framework in Section 2.3. In Section 2.4, we then review the framework of static games and its solution concept. Finally, we discuss the stochastic game framework in Section 2.5.

2.1 Agent Model

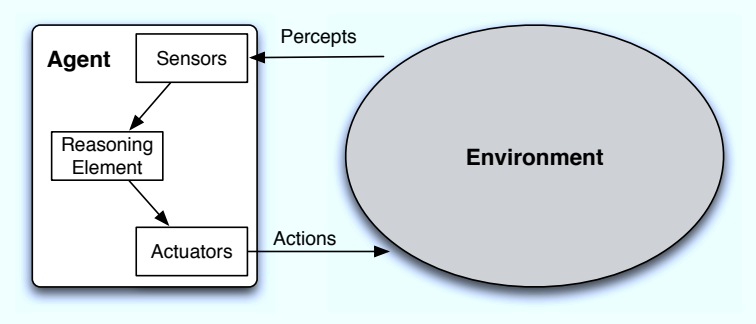


Figure 2.1. An agent model

An agent is an autonomous entity that has three key components: perception, reasoning, and action. These three components operate and interact with some environment as shown graphically in Figure 2.1. An agent receives a percept from the environment through its sensors, reasons what action to take based on its observation, and then performs the action through its actuators, which in turn affects the environment. The frameworks we describe in this chapter define a specific structure for the environment: what actions are available to the agent in the environment, how it is affected by the agent's actions, what percepts the agent can receive from the environment, and whether other agents are involved. Markov decision processes (MDPs) provide a model for the basic agent framework of Figure 2.1, where a single agent interacts with the environment.

The learning, which is the focus of this work, is used to improve the reasoning ability of choosing optimal actions based on its percepts. One of the main reasons for learning is that an agent does not know the details of the environment. The agent only receives information about the environment through its interaction, that is, by selecting actions and

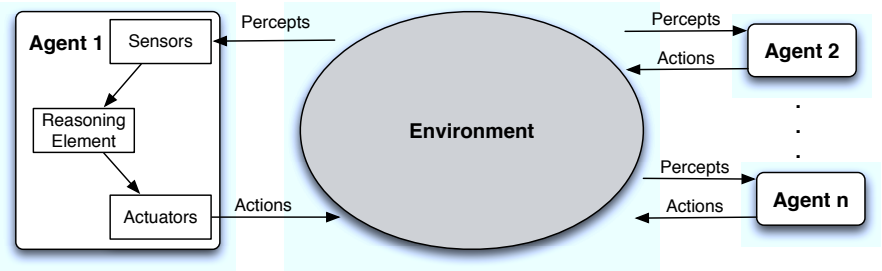


Figure 2.2. A multi-agent scenario: multiple agents all distinguished from their environment

observing their effects through its perceptual inputs. However, to learn effectively, the agent needs some feedbacks for its actions, which is the agent's reward. This reward depends on the state of the environment and the agent's action. The reasoning element of the agent contains the learning process that repeatedly interacts with the environment with the goal of maximizing the rewards it receives over time.

In this thesis, we are interested in learning in multi-agent settings. Figure 2.2 depicts a multi-agent scenario graphically. Instead of a single agent perceiving, reasoning, and acting in an environment, there are multiple complete agents. These agents also receive perceptions, reason, and act on the environment. Additionally, they may be learning agents as well, adapting their actions to maximize their own reward signals over time. Stochastic games or DEC-POMDPs correspond to the full multi-agent framework depicted in Figure 2.2. We now review the models of MDPs, DEC-POMDPs, and Stochastic games in turn.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) are the foundation for much of the research in the single agent control learning. They provides a formal framework for modeling single-agent decision-making problems with uncertainty. In this section, we will review the MDP model and its corresponding solution concepts. Reinforcement learning allows an agent

to learn the solution without knowing the details of a MDP. As our multi-agent learning algorithms are extensively built on reinforcement learning, we will briefly describe some MDP-based learning algorithms. Since, in this thesis, we are interested in learning with limited observability. In this section, we also discuss the learning in partial observable environments.

2.2.1 Definition

Definition 1. *A* **Markov decision process** *is defined by a tuple* $\langle S, A, T, R \rangle$*, where*

- S *is a set of states,*
- A *is a set of actions,*
- $T : S \times A \times S \rightarrow [0, 1]$ *is a transition function,*
- $R : S \times A \rightarrow \Re$ *is a reward function.*

The transition function defines a probability distribution over next states as a function of the current state and the agent's action. The reward function defines the reward received after selecting an action in the given state. Markov decision processes are called such because both their transition functions and reward functions satisfy the Markov Property, that is, the next state and the reward solely depend on the current state and action, and not on the history of states and actions. An agent interacts with an MDP environment by alternating between perception and action. The agent observes the state s^t at time t, and selects an action a^t. The agent then receives the reward $r^t = R(s^t, a^t)$, and observes the next state, $s^{t+1}$with the probability specified by the transition function $T(s^{t+1}|s^t, a^t)$. A sequence

$$s^0, a^0, r^0, s^1, a^1, r^1, \ldots s^t, a^t, r^t, \ldots$$

refers to a single execution trace of an agent in a MDP environment.

The MDP framework is a single-agent formalization of the agent environment. The agent's perception is the current state of the environment from the set S. The agent's

reasoning process is responsible for selecting an action from the set A in a given state. This closes the loop of agent perception, reasoning, and action. The learning process processes the observed next state and reward for taking an action in a state to improve the reasoning performance.

2.2.2 Solution Concept

The core problem of MDPs is to find an optimal *policy* for an agent. A policy $\pi : S \rightarrow A$ is a mapping function that specifies an action $\pi(s) \in A$ in each state $s \in S$. Once a Markov decision process is combined with a policy in this way, this fixes the action for each state and the resulting combination behaves like a Markov chain. Note that the policy defined here for MDPs is deterministic (always choosing a particular action for each state). We will define stochastic policies later, which are important for stochastic games.

An optimal policy for an MDP maximizes some function of the rewards received by executing the policy over a potentially infinite horizon. Typically, there are two types of functions: *discounted reward* and *average reward*.

Discounted Reward

In the discounted reward formulation, immediate reward is preferred over future reward. Specifically, the value of a policy π starting at state s, with a discount factor $\gamma \in [0, 1)$, is,

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbf{E}\{r^t | s^0 = s, \pi\},$$

where $\mathbf{E}\{r^t | s^0 = s, \pi\}$ is the expected reward received at time t given the initial state is s and the agent follows the policy π. V^{π} is called the policy's state value function. This formulation is similar to the economic principle of interest and investment, where utility now is traded against larger future utility. It can also be understood as describing the possibility that the process itself will terminate after any step with probability $gamma$, after which no additional reward can be accumulated. Another reason for considering discounted reward is that it leads to a finite expected reward and simplifies the mathematics.

Using this reward formulation, the goal for an agent is to find an optimal policy $\pi*$ that maximizes the discounted future reward for all states. If we know the state transition function T and the reward function R, we then can calculate the optimal policy using a standard family of algorithms, e.g., value iteration [14] and policy iteration [37]. In this thesis, we are more concerned about how to learn the optimal policy if we do not know the transition function and the reward function.

Average Reward

For many open systems that runs for a very long time, we are usually more interested in maximizing the average reward over the time, instead of the discounted reward. In this formulation, the value of a policy is defined relative to the average expected reward per time step under the policy. Mathematically, the value of a policy π at state s is,

$$V^{\pi}(s) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \mathbf{E}\{r^t | s^0 = s, \pi\},$$

A common assumption, which usually accompanies examinations of this reward formulation, is that the MDP is *unichain*. An MDP is unichain if and only if, for all policies, there exists an ergodic set of states (i.e., any state in the set can be reached with non-zero probability from any other state in the set), and all states outside this set are transient (i.e., after some finite point in time it will never be visited again). This assumption forces the value of a policy to be independent of the initial state. From any initial state, the policy is guaranteed to end up in the ergodic class, and any state in the ergodic class must have the same average reward for a given policy.

2.2.3 Reinforcement Learning

When the transition function or reward function of an MDP is unknown, an agent can not directly compute the optimal policy and needs to learn it through interacting with the environment. Reinforcement learning (RL) [100] is a field concerned with such learning in MDP environments. A broad spectrum of single-agent RL algorithms exists, e.g.,

model-free methods based on online estimation of value functions [117, 75, 98, 12, 108], and model-learning methods that estimate a model, and then learn using model-based techniques [99, 68]. As our multi-agent learning algorithms are built on top of Q-learning [117], we will briefly describe Q-learning here.

The Q-learning algorithm learns the optimal state-action value function. The state-action function (Q-value function) $Q^\pi : S \times A \rightarrow \Re$ defines the expected discounted reward of choosing a particular action from a particular state and then following the policy π. Formally, $Q^\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t \mathbf{E}\{r^{t+k} | s^k = s, a^k = a, \pi\}$. The optimal Q-value function Q^* satisfies the Bellman optimality equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a' \in A} Q^*(s',a'). \tag{2.1}$$

This equation states that the optimal value of taking a in u is the expected immediate reward plus the expected (discounted) optimal value attainable from the next state.

The policy is deterministic and picks the action with the highest Q-value for every state:

$$\pi(s) = \arg\max_{a \in A} Q(s,a). \tag{2.2}$$

The agent can achieve the learning goal by first computing Q^* and then choosing actions by the policy, which is optimal (i.e., maximizes the expected reward) when applied to Q^*.

Since the transition function is unknown, Q-learning turns Equation 2.1 into an iterative approximation procedure. The current estimate of Q^* is updated using estimated samples of the right-hand side of Equation 2.1. These samples are computed using actual experience interacting with the environment, in the form of the observed next state s^{k+1} and rewards r^{k+1} after taking action a^k in state s^k:

$$Q(s^k,a^k) \leftarrow Q(s^k,a^k) + \alpha^k [r^{k+1} + \gamma \max_{a^{k+1} \in A} Q(s^{k+1},a^{k+1}) - Q(s^k,a^k)]. \tag{2.3}$$

Since its update rule does not require knowledge about the transition and reward functions, Q-learning is model-free. The learning rate $\alpha^k \in (0, 1]$ specifies how far the current estimate $Q(s^k, a^k)$ is adjusted towards the update target $r^{k+1} + \gamma \max_{a^{k+1} \in A} Q(s^{k+1}, a^{k+1})$. The learning rate is typically time-varying, decreasing with time. Separate learning rates may be used for each state-action pair. The expression inside the square brackets is the temporal difference, i.e., the difference between estimates of $Q(s^k, a^k)$ at two successive time steps, $k + 1$ and k.

The sequence Q^k provably converges to Q^* under the following conditions [117, 41, 110]:

- Explicit, distinct values of the Q-function are stored and updated for each state-action pair.
- The time series of learning rates used for each state-action pair sums to infinity, whereas the sum of its squares is finite.
- The agent keeps trying all actions in all states with nonzero probability.

The third condition means that the agent must sometimes explore other actions than those determined by the current policy. To achieve this, one approach is to choose at each step a random action with probability $\epsilon \in (0, 1)$ and the greedy action with probability $(1 - \epsilon)$. This approach is called ϵ-greedy exploration. Another approach is to use the Boltzmann exploration strategy, which in state s selects action a with probability:

$$\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

where $\tau > 0$ is the temperature that controls the randomness of the exploration. When $\tau \to 0$, this is equivalent with the policy specified by Equation 2.2. When $\tau \to \infty$, action selection is purely random. For $\tau \in (0, \infty)$, higher-valued actions have a greater chance of being selected than lower-valued ones.

2.2.4 Partially Observable Environments

In many real-world environments, it will not be possible for an agent to have perfect and complete perception of the state of the environment. Unfortunately, complete observability is necessary for learning methods based on MDPs. In this section, we consider the learning case in which the agent makes observations of the state of the environment, but these observations may be noisy and provide incomplete information. We will first review the model of *partially observable Markov decision processes* (POMDP), an extension of the basic MDP framework for dealing with partially observability, and then discuss the learning in partially observable environments.

POMDP

Definition 2. *A* **partially observable Markov decision process** *is defined by a tuple* $\langle S, A, Z, T, O, R, b_0 \rangle$*, where*

- S *is a set of states,*
- A *is a set of actions,*
- Z *is a set of observations,*
- $T : S \times A \times S \to [0, 1]$ *is a transition function, where* $T(s'|s, a)$ *is the probability of transiting to the next state* $s' \in S$ *after a action* $a \in A$ *is taken by an agent in state* $s \in S$*,*
- $O : S \times A \times Z \to [0, 1]$ *is the observation function, where* $O(z|s, a)$ *denotes the probability of perceiving observation* z *when executing action* a *and arriving in state* s*,*
- $R : S \times A \to \Re$ *is a reward function, where* $R(s, a)$ *is the reward that an agent receives for taking action* a *in state* s*,*

- b_0 *is the initial state distribution, where* $b_0(s)$ *denotes the probability of starting in state* s*.*

A POMDP is really just an MDP, because the underlying dynamics of the POMDP are still Markovian. We have a set of states, a set of actions, transitions and immediate rewards. The actions' effects on the state in a POMDP is exactly the same as in an MDP. The only difference is in whether or not we can observe the current state of the process. In a POMDP, we add a set of observations Z to the model. So instead of directly observing the current state, the state gives us an observation which provides a hint about what state it is in. The observations can be probabilistic, so we need to also specify the observation model O. This observation model simply tells us the probability of each observation for each state in the model.

However, adding partial observability to an MDP is not a trivial addition. Since we have no direct access to the current state, selecting actions based on the current state (as in a MDP) is no longer valid. An agent has to act based on the history of executed actions and perceived observations. Hence, a policy in POMDP is defined as a mapping from action-observation histories to actions. Since the number of possible histories grows exponentially in the number of time steps, many POMDP algorithms use the concept of belief. Formally, the belief b is the probability distribution over the current states, where $b(s)$ denotes the probability that the state is s at the current time step. The belief update for the next time step can be computed from the belief at the current time step: given the action a at the current time step and the observation z at the next time step, the updated belief b_z^a for the next time step is obtained by

$$b_z^a(s') = O(z|s', a) \sum_s T(s'|s, a)b(s)/P(z|b, a),$$

where $P(z|b, a) = \sum_{s' \in S} O(z|s', a) \sum_s T(s'|s, a)b(s)$.

Hence, the belief serves as a sufficient statistic for fully summarizing histories, and the policy can be equivalently defined as a mapping from beliefs to actions. Using beliefs, we can view POMDPs as belief-state MDPs, and the value function of an optimal policy satisfies the Bellman equation

$$V^*(b) = \max_a [\sum_s b(s)R(s,a) + \gamma \sum_{s',z} T(s'|s,a)O(z|s',a)V^*(b_z^a)].$$

Learning in Partially Observable Environments

Approaches to learning in partially observable environments can be classified into two categories: learning with memory or without memory. Learning approaches with memory use previous actions and observations to disambiguate the current state. Such approaches build an internal representation of the state of environment by combining previous observations and even actions. Several different forms of internal representations have been used. As discussed in previous section, one representation is using the belief state to summarizing the history of observations and actions. Approaches [26, 64] using this representation employ some techniques (e.g., hidden Markov model techniques) to learn a model of the environment, including the hidden state, and then use POMDP algorithms to find a policy mapping belief states into action. Instead of using the whole memory, another representation [54, 65] is to use finite history windows of observations and actions to restore the Markov property. Recurrent neural networks [54] are also used to learn Q-values to retain "history observations".

A common drawback of all the above approaches to learning with memory is computationally expensive and can require a large amount of data. For example, finite-history-window approaches result in policy search spaces exponentially increasing with the window size, and approaches based on belief states can result in a continuous state space. Another drawback of these approaches is always based on strong assumptions about the environment. For example, it is assumed that the number of states is known in advance. Learning approaches without memory are often preferred in practical applications.

The most naive approach to learning without memory is to ignore partial observability and treat the observations as if they were the states of the environment and try to learn to behave. So the policy π is defined as a mapping from the immediate observation to an action. The resulting problem is not Markovian, and single-agent reinforcement learning algorithms, including Q-learning, cannot be guaranteed to converge. But small breaches of the Markov requirement are usually well handled by Q-learning.

However, the general hope that the performance of reinforcement learning algorithms will degrade gracefully as the degree of non-Markovianness is increased in a given decision problem is unfounded, because it is easy to construct simple environments where failure to distinguish between just two states can lead to an arbitrary high absolute loss in performance [93]. Singh, Jaakkola, and Jordan [93] showed that, in a POMDP, the best stationary stochastic policy could be arbitrary better than the best stationary deterministic policy (mapping an observation to an action). Stochastic policies are mappings from observations to probability distributions over actions. In this paper, we will use stochastic policies to deal with partial observability.

2.3 Decentralized Markov Decision Processes

In this thesis, we are interested in learning in cooperative multi-agent systems, where a group of agents work together to optimize the global performance. In this section, we will first review the framework of decentralized partially observable Markov decision processes (DEC-POMDP) to model the sequential decision-making problem in cooperative multi-agent systems. With this framework, we will also discuss how multi-agent learning is used to develop policies for agents in cooperative multi-agent systems.

2.3.1 Definition

The DEC-POMDP system starts with some initial state. The agents select actions based on their observations. The system then moves to a new random state whose distribution

depends on the previous state and the joint action chosen by the agents. The procedure is repeated at the new state and continues for a finite or infinite number of horizons. Formally,

Definition 3. *An n-agent DEC-POMDP is defined by a tuple $\langle S, A, T, Z, O, R, h \rangle$, where*

- *S is a set of states, with distinguished initial state s_0.*
- *$A = A_1 \times \cdots \times A_n$ is a set of joint actions, where A_i is the action set for agent i.*
- *$T : S \times A \times S \rightarrow [0, 1]$ is the transition function. $T(s'|s, a)$ is the probability of transiting to the next state s' after a joint action $a \in A$ is taken by agents in state s.*
- *$Z = Z_1 \times \cdots \times Z_n$ is a set of joint observations, where Z_i is the observation set of agent i.*
- *$O : S \times A \times Z \rightarrow [0, 1]$ is the observation function, where $O(z|s, a)$ denotes the probability of perceiving joint observation z after executing joint action a and arriving in state s.*
- *$R : S \times A \rightarrow \Re$ is the reward function. $R(s, a)$ is the reward for taking action $a \in A$ in state $s \in S$.*
- *If the DEC-POMDP has a finite horizon, that horizon is represented by a positive integer h.*

This framework was first proposed by Bernstein et al. [15]. DEC-POMDP [15] is a natural extension of POMDP to multi-agent settings by allowing multiple agents to collaboratively seeking to maximize a global performance. Each agent has an explicit action set. Instead of a single agent, there are multiple agents, whose joint action and the current state determine the distribution of the next state and rewards to the agents.

In DEC-POMDPs, agents together may not fully observe the system state (so we have only partial observability). We can define a generalization of MDP problems by requiring joint observability. A DEC-POMDP is jointly fully observable if the joint observation made

by the agents together fully determines the current state, that is, if, $\forall z \in Z \forall s \in S \forall a \in A$, $O(z|s,a) > 0$ then $P(s|z) = 1$. A DEC-POMDP that is jointly fully observable is called a decentralized Markov decision process (DEC-MDP).

2.3.2 Solution Concept

As with POMDPs, since an agent has no direct access to the current state in DEC-POMDPs or DEC-MDPs, selecting actions based on the current state (as in a MDP) is no longer valid. An agent needs act based on perceived observations. As with POMDPs, a local policy for agent i, $\pi_i : Z_i* \rightarrow A_i$, can be defined as a mapping from local histories of observations $\bar{(}z_i) = z_{i1}, \ldots, z_{ih}$ over Z_i to actions in A_i. In this way, the size of the policy space of each agent increases exponentially with the number of perceived observations. To promote the learning efficiency in DEC-POMDP environment, in this thesis, we use memory-less policies that maps an immediate observation to an action. As discussed in Section 2.2.4, stochastic policies can cope with the uncertainty of observations in certain degree and perform better than deterministic policies in partial observable environment. In this thesis, we define a policy $\pi : Z_i \times A_i \rightarrow [0,1]$ for agent i as a mapping of an observation $z_i \in Z$ to a probability distribution over actions A_i. We use $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ to refer to a joint policy for all the agents, with π_i being agent i's policy within that joint policy. Solving a DEC-MDP means finding a joint policy for agents that maximizes the expected total reward, which can be formulated by the following ways.

For a finite-horizon DEC-POMDP, the agents act for a fixed number of steps, which is called the horizon and denoted by h. The value of a joint policy π for a finite-horizon DEC-POMDP at state $s \in S$ is

$$V^{\pi}(s) = \sum_{t=0}^{h-1} \mathbf{E}\{r^t | s^0 = s, \pi\},$$

where $\mathbf{E}\{r^t | s^0 = s, \pi\}$ is the expected reward received at time t given the initial state is s and the agents follow the joint policy π.

For a infinite-horizon DEC-POMDP where the agents operate over an unbounded number of time steps, as with MDPs, the expected total reward formulations of discounted reward and average reward also can be applied to DEC-POMDPs to quantify the value of a joint policy. In the discounted reward framework, the value of the joint policy π at state $s \in S$, with discount factor γ, is

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbf{E}\{r^t | s^0 = s, \pi\},$$

Similarly, the average reward formulation in stochastic games is defined as,

$$V^{\pi}(s) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \mathbf{E}\{r^t | s^0 = s, \pi\}.$$

2.3.3 Learning in DEC-POMDP

For many complex cooperative multi-agent systems, it is very expensive, time-costly, or even not possible to obtain an accurate transition model or reward model of DEC-POMDP. This is especially true for applications operating in open environments where the environmental characteristics are not known a priori and may evolve over time. Learning provides an potential approach to developing policies for agents in cooperative systems.

Credit Assignment

To learn a policy using experience through interacting with the environment, an agent needs some reward feedbacks for their actions. In DEC-POMDPs, one potential way is to provide the global reward signal based on the joint action and the system state, which is usually the global performance measure of the system. However, for many large-scale multi-agent systems, it is usually infeasible to calculate the global performance measure in a real-time fashion. Therefore, local reward signals are needed for agent i to learn effectively, which can be efficiently computed and should be sufficient tailored to their behav-

iors. The problem of designing such reward signals is called multi-agent *credit assignment*. Chapter 3 Section 3.1 surveys related work in multi-agent credit assignment.

In this thesis, we assume that cooperative multi-agent systems use a credit assignment approach where as the local performance of an agent improves, given that the other agents use fixed policies, the global performance also improves, that is, the local performance is positively related to the global performance. One of such approaches is to linearly factor the global reward, which is used in Chapter 7. Formally,

Definition 4. *An n-agent DEC-MDP is said to be* **linearly reward factored** *if there exists functions $R_1, \ldots, R_n$, where $R_i : S \times A \rightarrow \Re$ is the reward function for agent i, such that, $R(s,a) = \sum_{i=1}^{n} w_i R_i(s,a)$, where w_i is a positive weight.*

Decentralized Learning

With local reward signals, decentralized learning provides an approximate, potentially scalable approach for DEC-POMDP problems, where each agent learns its local policy based on local observations and local rewards. By ignoring the actions and rewards of the other agents, this approach results in exponential storage and computational savings in the policy space and the value function.

The *independent learners* approach [27] employs Q-learning for each agent to learn its local policy. Although each agent has its action set, local observations, and local reward signals, its learning environment is not a MDP. Because the actions of the other agents are ignored in the representation of the Q-functions, and these agents also change their behavior while learning, the system becomes non-stationary from the perspective of an individual agent. As a result, the standard convergence proof for Q-learning does not hold anymore. Despite the lack of guaranteed convergence, this method has been applied successfully in multiple cases [101, 86, 24].

Although small breaches of the Markov requirement are well handled by Q-learning, there are many applications, including our motivating example described in detail in Chap-

ter 4, where the non-stationary environment causes Q-learning to oscillate and perform badly. In fact, decentralized learning treats DEC-POMDPs as partially observable stochastic games (POSG), where each agent can have its own specific reward function, and uses the equilibrium solution of POSG to approximate the optimal solution of DEC-POMDPs. Therefore, new techniques needs to be developed to learn effective policies in POSG. Chapter 3 will review some state-of-the-art multi-agent learning algorithms. Chapter 5 will present our multi-agent learning algorithm for stochastic games.

However, the field of multi-agent learning is still young and current multi-agent reinforcement learning algorithms, including our algorithms, still do not have theoretical convergence guarantees in POSG. In addition, using local reward signals may generate the tragedy of the commons problem, that is, greedy policies at agents can harm the global performance. Furthermore, as shown in Chapter 4, even with only using local observations, the policy space still remains large for many complex applications. As a result, decentralized learning may converge slowly, converge to inferior policies, or even diverge in DEC-POMDP problems, which is one major motivation for this thesis. Chapter 6 and 7 will present a supervision framework that employs low-overhead organizational control to coordinate decentralized learning, which improves multi-agent reinforcement learning's speed, quality, and likelihood of convergence of partial observable stochastic games for complex DEC-POMDP problems.

2.4 Static Games

As discussed in previous section, learning in DEC-POMDPs is often converted to learning in POSG. Before introducing POSG or stochastic games (described in next section), we want to review a simpler multi-agent framework, static or strategic games. Static games were first examined in the field of game theory to specifically model strategic interactions of multiple decision makers. Because of their simplicity of modeling interactions among multiple agents, static games have been increasingly adopted as a framework to formally

(a) Matching Pennies

$$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

(b) Coordination Game

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

(c) Rock-Paper-Scissors

$$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

(d) Shapley's Game

$$R_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 2.3. Examples of static games

analyze multi-agent learning algorithms (as we will show in Chapter 5). In this section, we will briefly review this framework and its solution concept, Nash equilibrium, and discuss the learning in this framework.

2.4.1 Definition

A static or strategic game [115, 73] is one in which all agents make decisions (or select a strategy) simultaneously, without knowledge of the strategies that are being chosen by other players. Even though the decisions may be made at different points in time, the game is simultaneous because each player has no information about the decisions of others; thus, it is as if the decisions are made simultaneously. Formally,

Definition 5. *A n-player* **static game** *is a tuple $(A_1, \ldots, A_n, R_1, \ldots, R_n)$, where*

- *A_i is the set of actions available to player i (and $A = A_1 \times \cdots \times A_n$ is the joint action space),*
- *and $R_i : A \to \Re$ is the payoff or reward function of player i.*

Each player select an action from their available set and receives a payoff that depends on all players actions (the joint action). Static games are represented by the normal form, where the payoff functions can be written as n-dimensional matrices. The actions then correspond to specifying the value of a particular dimension, and the joint actions correspond to particular entries in the payoff matrices. Figure 2.3 contains a number of example static games. As shown in the figure, the normal form only specifies an game's payoff functions using matrices and not the individual players' action sets, which are just assumed to be indices into the payoff matrices.

Static games can be classified according to the structure of their payoff functions. Two common classes of games are *team games* and *zero-sum games*. In team games (e.g., coordination game in Figure 2.3(b)), all agents have the same payoff function, so a joint action in the best interest of one agent is in the best interest of all the agents. In zero-sum games, there are two agents, and one's reward is always the negative of the other. The games (a) and (c) in Figure 2.3 are examples of such a game. The term *general-sum games* is used to refer to all types of games, including zero-sum games and non-zero-sum games. Note that Shapley game in Figure 2.3(d) is neither team games nor zero-sum.

2.4.2 Solution Concept

The goal of an agent or player in a static game is to find a strategy that maximizes its expected payoff. A *pure strategy* is one that deterministically selects a single action. However, pure strategies in static games can potentially be exploited. For example, in matching pennies shown in Figure 2.3. If one agent plays either action deterministically, then the other player can guarantee to win by playing the appropriate action. Therefore, mixed strategies are often more interesting in static games. A *mixed strategy* for player i, $\pi_i : A_i \rightarrow [0, 1]$, specifies a probability distribution over actions, where $\pi_i(a_i)$ is the probability of choosing action $a_i \in A_i$. These are the strategies we focus on. We use $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ to refer to a joint strategy for all of the players. With a joint strategy π,

the expected payoff of player i will be

$$V_i(\pi) = \sum_{a \in A} R_i(a) \prod_{i=1}^{n} \pi(a_i).$$

In MDPs, a solution is defined as the policy with the highest value according to some reward formulation, such as discounted reward or average reward. In static games, no single optimal strategy exists. A strategy can only be evaluated if the other players strategies are known. This can be illustrated in the game of matching pennies (Figure 2.3 (a)). In this game, if player 2 is going to play the first action, then the optimal strategy of player 1 is to play the first action, but if player 2 is going to play the second action, then the optimal strategy of player 1 is to play the second action. Therefore, one player can have optimal strategies only given the other players' strategies, which is called *best-response* strategies.

Definition 6. *Given a joint strategy π_{-i} of the other players, the* **best-response function** *$BR_i(\pi_{-i})$ for player i is the set of all strategies that are optimal. Formally, $\pi_i^* \in BR_i(\pi_{-i})$ if and only if $V_i(\langle \pi_i^*, \pi_{-i}) \geq V_i(\langle \pi_i, \pi_{-i}), \forall \pi_i \in \Pi_i$, where Π_i is the set of all mixed strategies for player i.*

In static games, one common solution is a joint strategy where every player's strategy is a best response for the other players' strategies so that no player can improve its expected payoff by unilaterally changing its own strategy. This solution is called a *Nash Equilibrium* [71].

Definition 7. *A* **Nash Equilibrium** *in a n-player static game is a joint strategy $\pi = \pi_1 \times \cdots \times \pi_n$ with $\pi_i \in BR_i(\pi_{-i}), \forall i = 1, \ldots, n$.*

One appealing property of this solution is that every static game has at least one (possibly mixed) Nash equilibrium. Some games have multiple Nash equilibria. For example, as shown in Figure 2.3, the game of matching pennies has one Nash equilibrium where both players play two actions randomly, that is, $\pi_1 = \langle 0.5, 0.5 \rangle$ and $\pi_2 = \langle 0.5, 0.5 \rangle$, while the

coordination game has two Nash equilibria, both player deterministically playing the first action or the second action.

For zero-sum games, an equilibrium can be computed efficiently using linear programming. However, finding equilibria in two-player general-sum games requires a more difficult quadratic programming solution [60]. Beyond two-player equilibrium solutions are even more difficult to find. McKelvey and McLennan [66] survey a variety of techniques for computing equilibria in matrix games, including n-player general-sum matrix games.

2.4.3 Learning in Repeated Games

Since we are interested in learning, we will focus on agents repeatedly playing the same static game. In game theory, this is called a *repeated game*. Its main difference from a one-shot static game is that the agents can use some of the game iterations to gather information about the other agents' strategies or the reward functions, and make more informed decisions thereafter. As in MDPs, each agent in a repeated game has an explicit action set. But, unlike MDPs, the environment has no state. With imperfect information, each agent can only perceive the actions of the other agents, or maybe just its own reward. In such cases, the agent needs to learn its strategy through experience to maximize its expected payoff.

Learning in repeated games is one case of multi-agent learning. One challenging question is that if all agents concurrently learn their strategies in repeated games, will their strategies converge to a Nash equilibrium? The challenge is that, when multiple agents concurrently learn their strategies, the environment is non-stationary or Markovian from the perspective of an individual learning agent. Although repeated games is stateless, the reward function of an learning agent depends on the other players' strategies. If other players are changing or adapting their strategy during the learning, then the agent's reward function is not Markovian any more. As a result, single-agent reinforcement learning algorithms may not converge in repeated games. Therefore, learning in repeated games

requires new or significantly modified machine learning techniques. Chapter 3 will review some state-of-the-art multi-agent learning algorithms and Chapter 5 will present our multi-agent learning algorithms analyzed and evaluated in repeated games.

2.5 Stochastic Games

As MDPs for single-agent reinforcement learning, stochastic games are more interesting for studying multi-agent learning than POSG. In this section, we will focus on stochastic games. Stochastic games [87] are a superset of MDPs and static games, which can have multiple agents and multiple states. They were first introduced in the field of game theory, and have now become a formal framework for studying multi-agent reinforcement learning. In this section, we will briefly describe this framework and its solution concept, and discuss multi-agent learning in stochastic games.

2.5.1 Definition

A stochastic game is a dynamic game with probabilistic transitions played by one or more players. The game is played in a sequence of stages. At the beginning of each stage the game is in some state. The players select actions and each player receives a payoff that depends on the current state and the chosen actions. The game then moves to a new random state whose distribution depends on the previous state and the actions chosen by the players. The procedure is repeated at the new state and play continues for a finite or infinite number of stages. Formally,

Definition 8. *An n-agent* **stochastic game** *is defined by a tuple $\langle A_1, \ldots, A_n, S, T, R_1, \ldots, R_n, \rangle$, where*

- *A_i is the set of actions available to player i (and $A = A_1 \times \cdots \times A_n$ is the joint action space).*
- *S is a set of states.*

- $T : S \times A \times S \rightarrow \Re$ *is the transition function.* $T(s'|s, a)$ *is the probability of transiting to the next state* $s' \in S$ *after a joint action* $a \in A$ *is taken by agents in state* $s \in S$.

- $R_i : S \times A \rightarrow \Re$ *is the payoff or reward function of player* i. *Agent* i *receives an individual reward* $R_i(s, a)$ *for the joint action* $a \in A$ *in state* $s \in S$.

Essentially, stochastic games are a generalization of Markov decision processes to the multi-agent cases. Each agent has an explicit action set. Instead of a single agent, there are multiple agents, whose joint action and the current state determine the next state and rewards to the agents. Also, note that each agent has its own independent reward function. When all players have the same reward function, such stochastic games are called *multi-agent Markov decision processes* (MMDP) [20], or *team stochastic games*. MMDP or team games is used to model decision-making problems in fully cooperative multi-agent systems where all agents individually observe the state of the environment.

Stochastic games can also be thought of as an extension of the concept of static games to multiple states. Each stochastic game has a static game associated with each state. The immediate payoffs for player i at a particular state s are determined by the function $R_i(s, \cdot)$. After selecting actions and receiving their rewards from the static game, the players are transitioned to another state and associated static game, which is determined by their joint action. The same classification for static games can be used with stochastic games. *Team games* are ones where all the agents have the same reward function. *Zero-sum games* are two-player games where one player's reward is always the negative of the other's for all states and all joint actions. *General-sum games* refer to all types of reward structures.

2.5.2 Solution Concept

Stochastic games borrow solution concepts from both MDPs and matrix games. Like MDPs, the goal for player i in a stochastic game is to find a policy that maximizes its long-term reward. Since deterministic strategies can be exploited in static games, deterministic policies can also be exploited in stochastic games. Therefore, we cannot restricted

ourselves to deterministic policies as is common with the study of MDPs. Throughout this work, we consider the full space of stochastic policies. A *stochastic policy* for player i, π_i, is a mapping that defines the probability of selecting an action from a particular state. Formally, $\pi_i \in S \times A_i \rightarrow [0, 1]$, where $\sum_{a \in A_i} \pi(s, a) = 1, \forall s \in S$.

Here are some notations. We use $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ to refer to a joint policy for all the players, with π_i being player is policy within that joint policy. We use the notation Π_i to be the set of all possible stochastic policies available to player i, and $\Pi = \langle \Pi_1, \ldots, \Pi_n \rangle$ to be the set of joint policies of all the players. We also use the notation π_{-i} to refer to a particular joint policy of all of the players except player i. Finally, the notation $\langle \pi_i, \pi_{-i} \rangle$ refers to the joint policy where player i follows π_i while the other players follow their policy from π_{-i}.

The reward formulations of discounted reward and average reward also can be applied to stochastic games to quantify the value of a joint policy to each player. In the discounted reward framework, the value of the joint policy π to player i at state $s \in S$, with discount factor γ, is

$$V_i^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbf{E}\{r_i^t | s^0 = s, \pi\},$$

where $\mathbf{E}\{r_i^t | s^0 = s, \pi\}$ is the expected reward to player i received at time t given the initial state is s and the agents follow the joint policy π. Similarly, the average reward formulation in stochastic games is defined as,

$$V_i^{\pi}(s) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T} \mathbf{E}\{r_i^t | s^0 = s, \pi\}.$$

Notice that a policy for a player can only be evaluated in the context of all the other players' policies. This is the same difficulty faced by static games and the same solution concepts from static games can be applied to stochastic games. We can define the concept of best response as following.

Definition 9. *Given a joint policy π_{-i} of the other players, the best-response function $BR_i(\pi_{-i})$ for player i is the set of all policies that are optimal. Formally, $\pi_i^* \in BR_i(\pi_{-i})$ if and only if $V_i^{\langle \pi_i^*, \pi_{-i} \rangle}(s) \geq V_i^{\langle \pi_i, \pi_{-i} \rangle}(s), \forall s \in S$ and $\forall \pi_i \in \Pi_i$.*

We can also define Nash equilibrium using the concept of best response.

Definition 10. *A Nash Equilibrium in a n-player static game is a joint strategy $\pi = \pi_1 \times \cdots \times \pi_n$ with $\pi_i \in BR_i(\pi_{-i}), \forall i = 1, \ldots, n$.*

2.5.3 Learning in Stochastic Games

The goal of a learning agent in stochastic games is to learn a policy that maximizes its long-term reward. As in MDPs, each learning agent in stochastic games has a explicit set of actions and can observe the state of the environment, but usually initially does not know the transition function of the environment and its reward function. Learning agents may also observe the actions of the other agents. To learn the policy to maximize its long-term reward, an agent needs to estimate this long-term value. As with MDPs, one approach is to use to Q-learning to learn its Q-value function defined on joint actions using experience of interacting with the environment and other agents.

Consider situations where only one agent is learning its policy and all the other agents use fixing stationary policies. the resulting decision process for the learning agent is a Markov decision process. The MDP's states and the player's action set are the same as in the original stochastic game. The new transition function is composed of the stochastic game's transition function with the other players' policies. Similarly, the reward function of the agent is composed of its reward function in the stochastic game with the other players' policies. Therefore, if the other agents are stationary and not learning, the problem for the learning agent simply reduces to learn to act in an MDP and Q-learning allows it to learn the optimal policy that maximizes its long-term reward

If multiple agents are concurrently learning their policies in a stochastic game, the learning environment becomes non-stationary from the perspective of individual learning agents. As with repeated games, the reward function of a learning agent is not Markovian. In addition, in stochastic games, the transition function for an individual agent is also not Markovian any more, because it is defined on actions of other agents who are changing or

adapting their policies over the time. As a result, the basic assumption behind single-agent reinforcement learning techniques, including Q-learning, is violated in multi-agent learning. For this reason, multi-agent learning requires new or significantly modified learning algorithms.

As in static games, the evaluation of an agent's policy depends on the other agents' policies. The optimality of an individual policy is meaningful only in the context of the other agents' policies. Therefore, Nash equilibrium is a potential solution concept for multi-agent learning, although there are a lot of debates about whether this solution concept is appropriate [88]. The analysis of the dynamics of multi-agent learning in stochastic games is much more difficult than that in repeated games. The convergence to Nash equilibria (or other solution concepts) of multi-agent learning is still an open and challenging problem. Chapter 3 will review some state-of-the-art multi-agent learning algorithms. Chapter 5 will present our multi-agent learning algorithm for stochastic games.

2.6 Summary

In this chapter, we presented the framework of DEC-POMDPs as a general model of multi-agent interaction in cooperative systems and discussed multi-agent learning in this framework. To better understand DEC-POMDP and multi-agent reinforcement learning, we reviewed the subsumed framework of Markov decision processes, studied in the reinforcement learning community. We also introduced the key solution concepts and algorithms for MDPs: discounted and average reward, value functions, optimal policies, and simple reinforcement learning techniques. To deal with multi-agent learning in large-scale cooperative systems where the global reward signal is not available or not specifically tailored to individual agents' performance, local reward signals were introduced for agents to learn their policies. As a result, we converted the problem of learning in DEC-POMDPs to learning in partially observable stochastic games. Therefore, we also introduced stochastic games and static games, one special cases of stochastic games, which were studied in the

field of game theory. We presented an overview of the key solution concepts in static games and stochastic games: best-responses and Nash equilibria and discussed learning in these frameworks. In the next chapter, we will survey techniques for learning in cooperative multi-agent systems.

CHAPTER 3

PREVIOUS WORK

In this chapter we explore related work on learning in cooperative multi-agent systems. In cooperative multi-agent systems, all agents work together to optimize the global performance measure. As discussed in Section 2.3.3 of previous chapter, due to limited communication in many large-scale cooperate systems, it is usually infeasible to calculate the global performance measure in a real-time fashion to be used as learning feedbacks. Local reward signals are needed for agents to learn effectively, which can be efficiently computed and sufficiently tailored to individual agents' behaviors. The design of local reward signals is usually referred as the *credit assignment* problem. So we begin by discussing the credit assignment problem in multi-agent learning.

As discussed in previous chapter, stochastic games provide a formal framework for studying multi-agent learning algorithms. We will examine multi-agent reinforcement learning algorithms in this framework. We first review algorithms specifically targeted to fully cooperative scenarios. As fully cooperative systems can be converted to non-cooperative scenarios by using unequal-share credit assignment, we then also discuss multi-agent learning in the general stochastic game framework.

Improving the suitability of multi-agent reinforcement learning to problems of practical interest is an essential research step. The *scalability* is a key challenge for multi-agent reinforcement learning, which is also one of the main focuses of this thesis. In this chapter, we will explore approaches to scaling up multi-agent learning to large systems.

This chapter focuses on the literatures of multi-agent reinforcement learning where agents are concurrently learning. Other learning techniques for cooperative multi-agent systems, such as evolutionary algorithms and centralized learning, are surveyed in [74].

3.1 Multi-Agent Credit Assignment

Reinforcement learning has the credit assignment problem in both single-agent and multi-agent domains. In single-agent multi-step domains, the credit assignment problem is concerned with how an action taken at a particular time step affects the final outcome. This problem can be called *temporal credit assignment*. For example, if a player wins a checker game, temporal credit assignment deals with how each move made during the game contribute to her/his win. Many reinforcement learning algorithms have been derived to assign proper credits for state-action pairs, including Q-learning, Sarsa, and TD(λ) [117, 75, 98, 12, 108]. The goal of these algorithms is to have an agent' learning converge to the optimal policy.

In cooperative multi-agent settings, multiple agents are concurrently learning to optimize the global performance. In addition to the temporal credit assignment problem, we also need to deal with the *structural credit assignment* problem, which determines how a single agent's actions contributes to the system performance. In order for a reinforcement learning agent to learn properly in cooperative multi-agent domains, this credit assignment problem needs to be resolved and the agent needs to receive the appropriate reinforcement. As shown in [4], the temporal credit assignment problem in single-agent mutli-step settings is equivalent with the structural credit assignment problem in multi-agent single-step settings. This work also argues that it may be possible to view a multi-agent multi-step problem as only a structural credit assignment.

One straightforward and simple solution to the credit assignment problem in cooperative multi-agent systems is to directly provide the global performance measure as the reward signal for every learning agent in the system. In this way, each learner optimizes its

performance, equivalently optimizing the global performance of the system. However, in many practical large-scale cooperative systems, because of limited communication bandwidth and computational resources, it is infeasible to calculate the global performance measure in a real-time fashion. Even when its calculation is possible, it still may not be desirable to use the global reward for multi-agent learning in many situations. If some learning agents make major contributions to a cooperative task, it is usually helpful for improving the global system performance to specially reward those learners for their actions or punish others for laziness. As argued in [120], using the global reward signal does not scale well to increasingly difficult problems because the learners do not have sufficient feedback tailored to their own specific actions.

In contrast to using the global reward signal, local rewards can be designed and used to evaluate each agent's performance solely based on its individual behavior. Using local reward signals can discourage laziness and spur each agent to improve its individual performance. The drawback of using local rewards is that greedy behaviors may develop, which sometimes harms the cooperation among agents and degrades the global performance. As shown by experiments in [9, 10], using local rewards can lead to faster learning rates, but not necessarily to better system performance results than using the global reward. For example, in one problem (foraging), using local rewards produces better results, while, in another problem (soccer), using the global reward is better.

The work [62] argues that agents' concurrent learning processes can be improved by combining individual local reward signal with some social reward signals. One type of social reward, called *observational reinforcement,* is obtained by observing other agents and imitating their behaviors, which may help improve the overall team behavior by reproducing rare behaviors. An agent additionally receives another type of social reward, called *vicarious reinforcement,* whenever other agents are directly rewarded. The purpose of vicarious reinforcement is to spread individual rewards to other agents, and thus balance between local and global rewards. The work shows that a weighted combination of

these social reward signals with individual local reward signal produces better global performance results in a foraging application.

Another work [5] presents a learning technique, called "Q Updates with Immediate Counterfactual Rewards learning" (QUICR-learning), which uses agent-specific rewards that ensure fast convergence in multi-agent coordination domains. Rewards in QUICR-learning are both heavily agent-sensitive, making the learning task easier, and aligned with the system level goal, ensuring that agents receiving high rewards are helping the system as a whole. QUICR-learning uses standard temporal difference methods but because of its unique reward structure, provides significantly faster convergence than standard Q-learning in large multi-agent systems

A different approach [33] is taken for credit assignment in cooperative multi-agent learning. This approach assumes that the reward signal observed by each agent is a sum of the agent's direct contribution and some random Markov process that estimates the contributions of teammates. The agent may therefore employ a Kalman filter to separate the two components and compute the agent's true contribution to the global reward. The authors show that this true contribution component provides a better feedback for learning in simple cooperative multi-agent domains.

This thesis deals with large-scale, complex cooperative multi-agent systems, where it is usually infeasible or prohibitively expensive to calculate the global performance measure in real-time. To trade off the local learning performance and the global system performance, the reward signal used for each agent's learning usually implicitly combines the local reward and the social reward. Although this credit assignment approach may improve the multi-agent learning performance, it can inadvertently create non-cooperative multi-agent learning environments, where the dynamics of the learning is more challenging and complex (as discussed in next section). In addition, this unequal-share credit assignment can also create the problem of "tragedy of the commons", that is, increasing the reward of an agent may degrade the global performance, and does not solve the scalability issue of multi-

agent learning. This thesis is intended to address these challenging issues of multi-agent learning in complex cooperative multi-agent systems, as discussed in Chapter 6.

3.2 Multi-Agent Reinforcement Learning Algorithms

The central challenge for multi-agent learning is that each learner is adapting its behaviors in the context of other co-adapting learners. When applying single-agent learning to stationary environments (e.g., MDP problems), the agent experiments with different policies by interacting with the environment until discovering a globally optimal policy. In dynamic environments, the agent may at best try to keep up with the changes in the environment and constantly track the shifting optimal behavior. One simplistic approach to dealing with co-adaptation is to treat the other learners as part of a dynamic environment to which the given learner must adapt. This idea was used in early multi-agent learning literature [84, 85, 132]. However, as agents concurrently learn, they modify their behaviors, which in turn can ruin other agents' learned behaviors. As a result, the environment becomes non-stationary from the perspective of individual agents, which violates the basic assumptions behind most traditional machine learning techniques [82, 118]. For this reason, entirely new multi-agent learning algorithms may be required to deal with this issues.

A framework is needed to provide a formal foundation for generating, analyzing, and evaluating new multi-agent learning algorithms. As discussed in previous chapter, stochastic games offers such a framework for studying multi-agent learning [55]. In stochastic games, an important concept is that of Nash equilibrium, which is a joint strategy (one strategy for each agent) where no agent has any rational incentive (in terms of better reward) to unilaterally change its strategy away from the equilibrium. The formal analysis of many multi-agent learning algorithms focuses on the convergence to a Nash equilibrium.

In the remaining of this section, we will first survey multi-agent learning in fully cooperative scenarios (i.e. team games or MMDP), where all agents receives the same global reward signal (Section 3.2.1). In such cases, increasing one's reward implies increasing

everybody else's reward. Therefore, it is relatively straightforward to check that the multi-agent learning approach has converged to the globally optimal Nash equilibrium. Although this thesis focuses on multi-agent learning in cooperative multi-agent systems, our MARL paradigm advocates designing and using local reward signals for individual agents' learning. By using local rewards, increasing the reward of an agent may not necessarily result in increasing the reward of all its teammates. Such credit assignment can inadvertently convert cooperative scenarios to non-cooperative ones. Therefore, we are also interested in learning algorithms for general multi-agent settings. Section 3.2.2 covers multi-agent learning in general-sum stochastic games, where the relationship among reward signals received by learners is less clear.

3.2.1 Fully Cooperative Stochastic Games

In a fully cooperative stochastic game (or team games), the agents have the same reward function and the learning goal is to maximize the common discounted return. If a centralized controller were available, the task would reduce to a multi-agent Markov decision process (MMDP), the action space of which would be the joint action space of the stochastic game. The Team Q-learning algorithm [57] assumes that the optimal joint actions are unique (which is rarely the case). Then, if all the agents learn the common Q-function in parallel with Q-learning, they can learn the optimal joint policy and maximize their return.

The Distributed Q-learning algorithm [51] solves the fully cooperative multi-agent decision-making problem with limited computation. Each agent maintains a local policy and a local Q-function, depending only on its own action. The local Q-values are updated only when the update leads to an increase in the Q-value. This ensures that the local Q-value always captures the maximum of the joint-action Q-values. By using this algorithm, the local policies of the agents provably converge to an optimal joint policies in deterministic repeated games. The work [44, 45] points out possible flaws in this distributed Q-learning

approach when dealing with stochastic environments, and present a modified exploration strategy that improves cooperation among agents.

Joint Action Learners [27] (JAL) learn joint-action values and employ empirical models of the other agents' policies. They propose two benchmark games (climb and penalty) and show that, using Q-learning, the convergence to global optimum is not always achieved in these games even if each agent can immediately perceive the actions of all other agents in the environment. They then develop several heuristics to increase the learner's Q-values for the actions with high likelihood of getting good rewards given the models. Brafman and Tennenholtz [25] introduce a stochastic sampling technique that is guaranteed to converge to optimal Nash equilibria. The algorithm is polynomial in the number of actions of the agents, but it assumes a priori coordination of the agents's learning processes: the agents agree to a joint exploration phase of some length, then agree to a joint exploitation phase (where each agent settles on the behavior that yielded maximum reward).

Optimal Adaptive Learning (OAL) [116] is developed for multi-step team stochastic games, which is guaranteed to converge to optimal Nash equilibria if there are a finite number of actions and states. In OAL, virtual games are constructed on top of each stage game of the stochastic game. In these virtual games, optimal joint actions are rewarded with 1, and the rest of the joint actions with 0. An algorithm is introduced that, by biasing the agent towards recently selected optimal actions, guarantees convergence to a coordinated optimal joint action for the virtual game, and therefore to a coordinated joint action for the original stage game. This is the first algorithm guaranteed to find the global optimum in fully cooperative stochastic games. Unfortunately, the optimality guarantee comes at a cost in scalability: the number of virtual games that need to be solved is exponential in the number of agents.

3.2.2 General-Sum Stochastic Games

There are two broad classes of learning algorithms with very different explicit goals: equilibrium learners and best-response learners. Equilibrium learners explicitly seek to estimate and converge to their policy in one of the game's Nash equilibria. Best-response learners seek to directly learn and play a best-response to the other players' policies. Although not explicitly and directly seeking to converge to Nash equilibria, many best-response learning techniques are shown to converge to Nash equilibria in some limited settings. In this section, we review both classes of learning algorithms respectively.

Equilibrium Learners

There has been a line of research over the past decade in regards to the development of equilibrium learning algorithms, as well as determining their conditions for convergence. The Minimax-Q [55] algorithm extends the traditional Q-Learning algorithm for MDPs to zero-sum stochastic games. This algorithm provably converges to the stochastic game's equilibrium solution, assuming the other agent executes all of its actions infinitely often.

Nash-Q [38, 21, 39] extends the Minimax-Q algorithm to two-player general-sum stochastic games. The extension requires that each agent maintain Q values for all of the agents. Also, the linear programming solution used to find the equilibrium of zero-sum games is replaced with the quadratic programming solution for finding an equilibrium in two-player general-sum games. This algorithm is the first to address the complex problem of general-sum stochastic games. But the algorithm requires a number of very limiting assumptions. With the Nash-Q algorithm, agents do not learn just a single table of Q-values, but also tables for all other agents. This extra information is used later to approximate the actions of the other agents. An alternative approach [69] is proposed, where agents approximate the policies, rather than the tables of Q-values, of the other agents.

Friend-or-Foe-Q (FFQ) [56] is an equilibrium learner that extends Minimax-Q to include a small class of general-sum games. Motivated by the assumptions of Nash-Q, which

required that either the game be effectively zero-sum, so each intermediate game has a saddle point equilibrium, or the game was a team game, so each intermediate game has a global optimum. This extension handles both of these classes of games, as well as others, that do not by themselves fit under the Nash-Q assumptions. Like Minimax-Q, FFQ is guaranteed to converge to their policy in an equilibrium for the stochastic game.

Another equilibrium learning technique is Correlated-Q (CE-Q) [29] that seeks to learn to play according to an equilibrium by using the broader class of correlated equilibria. Nash equilibria are independent stochastic distributions over player's actions. Correlated equilibria allow for stochastic distributions over joint actions, where players do no randomize independently. CE-Q is more efficient than Nash-Q, since it does not require the complex quadratic programming Nash equilibrium solver.

The final equilibrium learning technique is AWESOME [28] that uses fictitious play, but monitors the other agents and, when it concludes that they are nonstationary, switches from the best-response in fictitious play to a centrally precomputed Nash equilibrium (hence the name: Adapt When Everyone is Stationary, Otherwise Move to Equilibrium). In repeated games, AWESOME is provably convergent.

Best-Response Learners

Best-response learning algorithms do not explicitly consider equilibria. Instead, they simply attempt to learn a best-response to the other player's current policies. A major consideration for looking at best-response learning is that agents are not always fully rational. Playing an equilibrium policy is only sensible when the other agents also play according to the equilibrium. When considering agents with limited perception, they may not be capable of learning or playing the equilibrium. Best-response algorithms have the possibility of both coping with limited teammates as well as exploiting limited opponents.

Q-Learning [117] is a single-agent learning algorithm specifically designed to find optimal policies in MDPs. In spite of its original intent, it has been widely used for multi-agent

learning [101, 86, 82, 27]. However, Q-learning traditionally cannot learn or play stochastic policies. This prevents Q-learners from converging to equilibria solutions for games only having mixed equilibria (e.g., zero-sum games). A particular value-based learning algorithm, called individual Q-learning [52], which extends Q-learning and uses stochastic approximation, can lead strategies to converging to Nash distributions almost surely in 2-player zero-sum games and 2-player team games. Evolutionary game theory has been linked to Q-learning and provides useful insights into the learning dynamics [19, 112, 111]. A learning algorithm, called Frequency Adjusted Q-learning (FAQ-learning), is proposed as a variation of Q-learning that complies with the prediction of the evolutionary model derived in [112, 111]. The convergence of FAQ-learning is analyzed in three types of games: Matching pennies, Prisoners' Dilemma and Battle of Sexes [42].

Another best-response learning algorithm is *Infinitesimal Gradient Ascent (IGA)* [92], which has one of the first theoretical results on convergence for a gradient-based multi-agent learning algorithm. The authors analyze the gradient ascent algorithm in two-player, two-action, general-sum repeated games by examining the dynamics of the strategies in the case of an infinitesimal step size. Its main conclusion is that, if both players use IGA, their average payoffs will converge in the limit to the expected payoffs for some Nash equilibrium. However, its notion of convergence is still weak. It is because, although the players' average payoffs converges, their strategies may not converge to a Nash equilibrium. As a result, their expected payoffs may vary greatly for different periods. To address this convergence problem, the work [23] introduce the WoLF principle ("Win or Learn Fast") to IGA and propose an algorithm called *WoLF-IGA*, which varies the learning rate from small and cautious values when winning, to large and aggressive values when losing to the other agents. The WoLF-IGA algorithm guarantees Nash convergence in two-player, two-action, general-sum repeated games.

The work [133] looks at gradient ascent using the evaluation criterion of regret and extends IGA beyond two-player, two-action games. A new algorithm, *GIGA (Generalized In-*

finitesimal Gradient Ascent), is proposed, which updates strategies using an unconstrained gradient, and then projects the resulting strategy vector back into the simplex of legal probability distributions. It is proved that GIGA has no-regret for online convex programming, a superclass containing normal-form games. Since GIGA is identical to IGA in two-player, two-action games, GIGA also has the weak form of convergence in this subclass of games. *GIGA-WoLF* [22] introduces the WoLF principle into GIGA to achieve GIGA's no-regret result and part of WoLF-IGA's convergence result. Another multi-agent learning algorithm is *Weighted Policy Learner (WPL)* [3], which uses a similar idea to the WoLF principle. WPL empirically outperforms both WoLF-IGA and GIGA-WoLF. Chapter 5 will present two multi-agent learning algorithms, one possessing the same convergence guarantee as WoLF-IGA, and another empirically outperforms state-of-the-art multi-agent learning algorithms.

3.3 Scaling up Multi-Agent Learning

Scalability is a problem for many learning techniques, but especially so for multi-agent learning. The dimensionality of the search space grows rapidly with the number and complexity of agent behaviors, the number of agents involved, and the size of the network of interactions between them. As shown in [129], even with only using local observations, the policy space still remains large for many complex applications. In addition, with unequal-share credit assignment, increasing the reward of an agent may not necessarily result in the system performance, which may generate the "tragedy of the commons problem", that is, greedy policies at agents can harm the global performance. With additional factors in realistic settings, such as a non-stationary environment, communication delay between agents, and partial observability, multi-agent learning in large-scale complex cooperative systems can be very slow, have inferior quality, and even diverge.

To improve the performance of multi-agent learning in complex systems, there several classes of approaches has been proposed. One kind of approaches, called *reward shaping*,

have been proposed, which gradually changes the reward function from favoring easier behaviors to favoring more complex ones based on those easy behaviors. The work [10] uses a shaped reinforcement reward function (earlier suggested by [63]) which depends on the number of partial steps fulfilled towards accomplishing the joint task. The author shows that using a shaped reward leads to similar results to using a local reward, but in a significantly shorter time.

One approach is to reduce the policy search space. TPOT-RL [96] reduced the state space by mapping states onto a limited number of action-dependent features. Another technique is hierarchical MARL [59], where the explicit task structure was used to restrict the space of policies. Each agent learned joint abstract action-values by communicating with others only the state of high-level subtasks, rather than primitive action they may perform. Learning techniques [31, 46] based on coordination graphs exploit the dependency structure between agents to decompose the global payoff function into a sum of local terms. The computation of the global value function is distributed by passing messages. However, this message passing results in heavy communication overhead for each value function update, which is not scalable for large agent networks.

Another approach is to employ pre-specified heuristics to guide the policy search. Heuristically Accelerated Minimax-Q (HAMMQ) [17] incorporated heuristics into the Minimax-Q algorithm to speed up its convergence rate. HAMMQ shared the convergence property with Minimax-Q [55]. However, HAMMQ was intended for use only in a two-agent configuration. Its authors used hand-coded domain heuristics, which did not capture the dynamics of other learning agents. Another work [102] used both local and global heuristics to accelerate the learning process in a decentralized multi-robot system. The local heuristic was derived from local information of an agent (i.e., robot), while the global heuristic was derived from the global data obtained from other agents. The global data needed to be exactly the same among agents. This consistency was maintained by broadcasting messages among all agents, which incurred heavy communication overhead and did not scale well.

In addition, this work was developed specifically for the multi-robot patrolling problem. A reinforcement learning based algorithm [67] was proposed for independent agents to learn both individual policies and when and how to coordinate. This algorithm exploited sparse interaction between agents to minimize the coupling of the learning processes for the different agents. However, the algorithm was described for only two-agent settings.

Multi-agent reinforcement learning (MARL) faces the *problem of reinvention*, that is, as agents are treated as separate subproblems, they usually separately discover and represent all aspects of the solution, even though optimally there may be a high degree of overlapping information among the policies of agents. Several techniques have been proposed to avoid this reinvention problem in order to improve the MARL performance. One technique is to share information among cooperative learning agents [101]. Several ways of sharing information have been studied: 1) sharing sensation, 2) sharing episodes, and 3) sharing learned policies. The author shows that (a) additional sensation from another agent is beneficial if it can be used efficiently, (b) sharing learned policies or episodes among agents speeds up learning at the cost of communication, and (c) for joint tasks, agents engaging in partnership can significantly outperform independent agents although they may learn slowly in the beginning. Another technique, called *imitation* [79], allows an agent to learn how to act well (perhaps optimally) by passively observing the actions of cooperative teachers or other more experienced agents in its environment. An alternative evolutionary approach approach, called *hypercube-based neuroevolution of augmenting topologies (HyperNEAT)*, is proposed to address this reinvention problem. HyperNEAT encodes the team as a pattern of related policies rather than as a set of individual agents. To capture this pattern, a policy geometry is introduced to describe the relationship between each agent's policy and its canonical geometric position within the team. Because policy geometry can encode variations of a shared skill across all of the policies it represents, the problem of reinvention is avoided. Furthermore, because the policy geometry of a particular team can

be sampled at any resolution, it acts as a heuristic for generating policies for teams of any size, producing a powerful new capability for multiagent learning.

3.4 Summary

Multi-agent learning is still a new field and most of its research challenges are still open to explore. With multi-agent learning, agents are concurrently learning their policies and adapting to each other. This co-adaptation of learners results in a non-stationary environment for an individual learning agent, which is a unique challenge not normally found in single-agent learning. In Chapter 5, we will present new multi-agent learning algorithms that are intended to address this challenge.

Unequal credit assignment can convert an ordinary cooperative scenario into a general-sum or non-cooperative scenario. Most state-of-the-art multi-agent learning algorithms focus on whether they converge or not to an equilibrium, and not on which equilibrium they converge to. As a result, in many cases, the learning may converge to inferior equilibria but not optima. In addition, scalability is still a key challenge for multi-agent systems to be applied to practical problems. Many state-of-the-art techniques to speeding up MARL are either restricted to specific domains or not scalable in large agent networks. In Chapter 6 and 7, we will present a supervision framework that employs low-overhead organizational control to coordinate decentralized learning, which improve multi-agent reinforcement learning's speed, quality, and likelihood of convergence in complex DEC-POMDP problems. Some techniques, such as TPOT-RL, that reduce the state space can be used together with our proposed framework for further speeding up MARL.

Part II

MULTI-AGENT LEARNING ALGORITHMS

CHAPTER 4

A MULTI-AGENT LEARNING APPROACH TO ONLINE DISTRIBUTED RESOURCE ALLOCATION

Learning is a key component of multi-agent systems (MAS), which allows an agent to adapt to the dynamics of other agents and the environment and improves the agent performance or the system performance (for cooperative MAS). The main purpose of this thesis is to develop MARL techniques that can scale up and be more easily applied to large complex MAS applications. This chapter is intended to demonstrate applicability and effectiveness of multi-agent learning for complex cooperative multi-agent domains where each agent has a limited view and can not access to the global reward signal in a real-time manner. Meanwhile, we also would like to investigate outstanding issues of applying MARL in complex applications. We design a gradient-based multi-agent learning algorithm that extends Q-learning to learn stochastic policies and apply it to optimize distributed resource allocation problem in cloud computing. The work of this chapter was published in IJCAI 2009 [129].

4.1 Introduction

As "Software as a service" becomes a popular business model, it is becoming increasingly difficult to build large cloud computing infrastructures that can host effectively the wide spread use of such services. *Shared clusters* built using commodity PCs or workstations offer a cost-effective solution for constructing such infrastructures. Unlike a dedicated cluster, where each computing node is dedicated to a single application, a shared cluster can run the number of applications significantly larger than the number of nodes, necessitating resource sharing among applications. Resource management approaches developed for shared clusters [8, 7, 113] are centralized, which limits the cluster scale.

To build larger shared computing infrastructures, one common model is to organize a set of shared clusters into a network and enables resource sharing across shared clusters. The resource allocation decision is now distributed to each shared cluster. Each cluster still uses a cluster-wide technique for managing its local resources. However, as task (also referred to applications services) allocation requests vary across clusters, an cluster may need to dynamically decide what tasks to allocated locally and where to forward unallocated tasks to cooperatively optimize the global utility of the whole system. To achieve scalability, each cluster has limited number of neighboring clusters that it interacts with.

We describe this decision problem as a distributed sequential resource allocation problem (DSRAP). We consider DSRAP is a novel and practical application for multi-agent learning. In DSRAP, each agent (referred to a cluster) has only a partial view of the whole system and does not have access to the system-level utility (because it is not directly measurable in real-time). All agents make decisions concurrently and autonomously. Each agent's decision depends not only on its local state but also on other agents' states and policies.

We use a multi-agent learning algorithm, called Fair Action Learning (FAL), which is a approximate variant of the Generalized Infinitesimal Gradient Ascent (GIGA) algorithm [133], for each agent to learn local decision policies. FAL is intended for limited observable environments and only requires the observation of the reward signals. To simplify the learning, we decomposes each agent's decisions into two connected learning problems: *local allocation problem* (deciding what tasks to be allocated locally) and *task routing problem* (deciding where to forwarded a task). To avoid poor initial policies during learning, heuristic strategies are developed to speed up the learning. The learning approach is tested in a network of simulated clusters and compared with a centralized greedy allocation approach, which is optimal in some cases. Experimental results show that our multi-agent learning works effectively and even outperforms the centralized approach in some cases. Although we discuss our approach in this particular problem, it can be more generally use-

ful in other online resource allocation problems, for example, when shared resources are storage devices in distributed file systems, documents in peer-to-peer information retrieval, or energy in sensor networks.

The rest of this chapter is structured as follows. Section 4.2 defines DSRAP. Section 4.3 introduces the Fair Action Learner algorithm. Section 4.4 presents decision-making processes of each agent and learning models for both decisions. Section 4.5 describe experiment design and analyzes experimental results. Related work is presented in Section 4.6. Finally, Section 4.7 concludes our work.

4.2 Problem Description

The runtime model of DSRAP is described as follows. Each agent receives tasks from either the external environment or a neighbor. At each time step, an agent makes decisions on what tasks are allocated locally and to which neighbors the tasks not allocated locally should be forwarded. Due to the task transfer time cost, there is communication delay between two agents. To reduce the communication overhead, the number of tasks an agent can transfer at each time step is limited. To allocated a task, an agent should have available resources to satisfy its resource requirements. When a task is allocated locally, the agent gains utility at each time step, which is specified by the task utility rate. If a task can not be allocated within its maximum waiting time, it will be removed from the system. If an allocated task is finished, all resources it occupies will be freed and available for future tasks. The main goal of DSRAP is to derive decision policies for each agent that maximize the average utility rate (AUR) of the whole system.

We denote a DSRAP with a tuple $\langle \mathscr{C}, \mathcal{A}, \mathcal{T}, \mathcal{B}, \mathcal{R} \rangle$, where

- $\mathscr{C} = \{C_1, \ldots, C_m\}$ is a set of shared clusters.
- $\mathcal{A} = \{a_{ij}\} \in \Re^{m \times m}$ is the adjacent matrix of clusters and each element a_{ij} is the task transfer time between cluster C_i and cluster C_j.

- $\mathcal{T} = \{t_1, \ldots, t_l\}$ is a set of task types.
- $\mathcal{B} = \{D_{ij}\}$ is the task arrival pattern and D_{ij} is the arrival distribution of tasks of type t_j at cluster C_i.
- $\mathcal{R} = \{R_1, \ldots, R_q\}$is a set of resource types (e.g., CPU and network) that each cluster provides.

Each cluster $C_i = \{n_{i1}, n_{i2}, \ldots, n_{ik}\}$ contains a set of computing nodes. Each computing node n_{ij} has a set of resources, represented as $\{\langle R_1, v_{ij1}\rangle, \ldots, \langle R_q, v_{ijq}\rangle\}$, where R_h ($h = 1, \ldots q$) is the resource type and $v_{ijh} \in \Re$ is the capacity of resource R_h on node n_{ij}. We assume there exist standards that quantify each type of resource. For example, we can quantify a fast CPU as 150 and a slow one with a half speed as 75.

A task type characterizes a set of tasks. A task type t_i is also denoted as a tuple $\langle D_i^s, D_i^u, D_i^w, D_i^{d_1}, \ldots, D_i^{d_q}\rangle$, where

- D_i^s is the task service time distribution
- D_i^u is the task utility rate (utility per time step) distribution
- D_i^w is the distribution of the task maximum waiting time before being allocated
- $D_i^{d_j}$ is the demand distribution of resource j of a task.

A task is denoted as a tuple $\langle t, u, w, d_1, \ldots, d_q\rangle$, where

- t is the task type.
- u is the utility rate of the task.
- w is the maximum waiting time before being allocated.
- d_i is the demand of resource $i = 1, \ldots, q$.

Based on the model of DSRAP developed above, the average utility rate of the whole system to be maximized can be defined as following:

$$AUR = \lim_{n\to\infty} \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{x\in T_i(C_j)} u(x)}{n} \tag{4.1}$$

where $T_i(C_j)$ is the set of tasks that allocated to cluster C_j at time i and $u(x)$ is the utility of task x. Note that, due to its partial view of the system, each individual cluster can not observe the system's AUR.

4.3 Fair Action Learning Algorithm

In the single-agent setting, reinforcement learning algorithms, such as Q-learning, learn optimal value functions and optimal policies in MDP environments when lookup tables are used to represent the state-action value function. However, in the multi-agent setting, due to the non-stationary environment (all agents are simultaneously learning their own policies), the usual conditions for single-agent RL algorithms' convergence to an optimal policy do not necessarily hold [27]. As a result, the learning of agents may diverge due to lack of synchronization. Several multi-agent reinforcement learning (MARL) algorithms have been developed to address this issue [133, 22], with convergence guarantee in specific classes of games with two agents.

Algorithm 1: Fair Action Learning (FAL) Algorithm

begin
  $r \leftarrow$ the cost for action a at state s;
  update Q-value function with $< s, a, r >$;
  $\bar{r} \leftarrow$ average reward $= \sum_{a\in A} \pi(s,a)Q(s,a)$;
  foreach *action* $a \in A$ **do**
    $\Delta(s,a) \leftarrow \zeta(Q(s,a) + \bar{r})$;
  end
  $\pi(s) \leftarrow limit(\pi(s) + \Delta(s))$;
end

To address DSRAP, we propose a multi-agent reinforcement learning algorithm, called Fair Action Learning (FAL). The FAL algorithm is a direct policy search technique and a variant of the GIGA algorithm [133] that approximates the policy gradient of each state-action pair with the difference of the expected Q-value on that state and its Q-value. Algorithm 1 describes its policy update rule, where ζ is the policy learning rate. FAL learns stochastic policies. As argued in [94], stochastic policies can work better than deterministic policies in partially observable environments (e.g., DSRAP), if both are limited to act based on the current percept. To improve the expected value for each state, FAL will increase the probability of actions that receive an expected reward above the current average. Therefore FAL will converge to a policy where, for each state, all actions receive the same expected reward and are fairly treated. (It is possible that FAL converges to a deterministic policy when an action is always more favorable than other actions). In a multi-agent setting, this property will help agents to converge to an equilibrium.

To normalize $\pi(s)$ such that it sums to 1, the *limit* function from GIGA [133] is applied with minor modifications so that every action is explored with minimum probability ϵ:

$$\pi(s) = limit(\pi(s)) = argmin_{x:valid(x)}|\pi(s) - x|$$

i.e., $limit(\pi(s))$ returns a valid policy that is closest to $\pi(s)$.

4.4 Learning Distributed Resource Allocation

Algorithm 2 shows the general decision-making process of each agent, which repeats at each time step. This algorithm uses two functions: *selectAndAllocate* and *chooseANeighborAndForward*. The first function selects and allocates a subset of received tasks to its local cluster to maximize its local utility. As the global utility is the sum of all local utilities, optimizing this function can potentially improve the system performance. The second function chooses a neighbor and forwards an unallocated task to maximize the allocation

Algorithm 2: General Decision-Making Algorithm

```
1  begin
2      TASKS ← set of tasks received in current time cycle;
3      ALLOCATED ← selectAndAllocate(TASKS);
4      TASKS ← TASKS \ ALLOCATED ;
5      foreach task t ∈ TASKS do
6          chooseANeighborAndForward(t) ;
7      end
8  end
```

probability of the task. This function aims to route tasks to unsaturated agents and balance the task load in the system.

4.4.1 Local Allocation Decision

Algorithm 3: *selectAndAllocate*(TASKS)

```
1  begin
2      ALLOCABLE ← getAllocable(TASKS);
3      ALLOCATED ← ∅ ;
4      while ALLOCABLE ≠ ∅ do
5          ALLOCABLE ← ALLOCABLE ∪ {VOID} ;
6          update current state s;
7          t ← task selected based on policy π1(s, ·);
8          if t = VOID then
9              ALLOCABLE ← ∅;
10         else
11             allocate(t);
12             ALLOCATED ← ALLOCATED ∪{t} ;
13             TASKS ← TASKS \{t} ;
14             ALLOCABLE ← getAllocable(TASKS);
15             learn(s, t);
16         end
17     end
18     return ALLOCATED;
19 end
```

Algorithm 3 shows the local allocation decision-making process. This algorithm incrementally selects and allocate tasks locally. It uses three functions: *getAllocable*, *allocate*, and *learn*. Function *getAllocable* filters *tasks* based on current local resource availability

and returns allocable tasks. Function *allocate* is responsible for allocating resources to the task and update local resource availability information. Function *learn* updates its allocation decision policy for selecting a task. Here we use π_1 to denote the local allocation policy. *VOID* is a unique, fake task with no resource requirements and zero utility rate. Selecting this task indicates that the process of selecting a subset of tasks to be allocated locally is finished.

Now we define the state space, the action space, and the reward function for learning this decision policy. A decision state $s = \langle s_t, s_c \rangle$ consists of two feature vectors s_t and s_c, describing the task set to be allocated and availability of various resources in a cluster respectively. As the task type of a task approximately represents information about the task, we use task types to characterize the task set to be allocated. The feature vector $s_t = \langle y_1, y_2, \ldots, y_m \rangle$, where each feature y_i corresponds to task type i and m is the number of task types. If the task set contains a task with type i, then $y_i = 1$. To represent s_c, we first categorize availability of each resource into multiple levels and then use combinations of levels of different resources as features. The value of a feature is the number of computing nodes in the cluster that have corresponding availability level for each resource.

An action of this decision is to select a task to allocate. So each task t corresponds to an action. In a real environment, it is not frequent to see two tasks that are exactly the same. To reduce the action space, the type of the task is used to approximately represent the task itself. Therefore, the action set is mapped to the set of task types. Then the binary feature vector s_t of an abstract state s determines available actions for state s. It is possible that one task set to be allocated may have several tasks with the same type. When such a task type is selected, the task of this type with the greatest utility rate will selected and allocated. The reward for allocating task t is the utility rate associated with t.

An agent receives tasks from both the external environment and its neighbors. Other agents' decision policies will affect task arrivals at the agent. As all agents concurrently learn their policies, the learning environment of each agent becomes non-stationary. We use

FAL algorithm to learn local allocation decision policies $\pi_1(s, a)$. As $\pi_1(s, a)$ is stochastic, the following rule is used to update Q-value function:

$$\begin{aligned} Q(s_n, a_n) \quad &\leftarrow (1-\alpha)Q(s_n, a_n) \quad + \\ &\alpha[r_n + \gamma \textstyle\sum_a \pi(s_{n+1}, a)Q(s_{n+1}, a)] \end{aligned}$$

This new update rule is just like that of Q-learning except that instead of the maximum over next state-action pair it uses the expected value under the current policy.

Accelerating the Learning Process

Even when using the approximated state space and action space developed above, the state-action space of each agent is still extremely large. Assume that a cluster has n computing nodes, m types of resources, and receives k types of tasks and availability of each resource is discretized into d levels, the size of the state-action space is $k2^k n^{d^m}$. In addition, any pure knowledge-free reinforcement learning exploration strategies could entail running arbitrarily poor initial policies, which should be avoided in the practical system. To address those issues, we proposed several heuristics to speed up learning. Policies are initialized with a greedy allocation algorithm, which allocates all tasks in an decreasing order of their utilities if resources permit. The learning is online and the ϵ-greedy strategy is used to ensure that each action will be explored with a minimum rate. To avoid unwanted system performance, we set a utilization threshold for each cluster. If the utilization of every resource is below this threshold, then the manager stops ϵ-greedy exploration and uses the greedy algorithm for exploration. In addition, rejecting too many tasks will degrade the system performance and thus we also limit the exploration rate of selecting *VOID* task.

4.4.2 Task Routing Decision

Task routing addresses the question: to which neighbor should an agent forward an unallocated task to get it to a unsaturated cluster before it expires? As each agent interacts with a limited number of neighbors, it may not know where are unsaturated clusters that

can be multiple hops away from it. An agent can learn to route tasks via interacting with its neighbors. The learning objective for task routing is to maximize the probability of each task to be allocated in the system.

The state s_x is defined by the characteristics of the task x that an agent is forwarding. More specifically, s_x can be represented by a feature vector $\langle t_x, w_x \rangle$, where t_x is the type of the task x and w_x is the remaining waiting time of the task x. An action j corresponds to choosing neighbor j for forwarding a task. The value function $Q_i(s_x, j)$ returns the expected probability that the task x will be allocated if an agent i forwards it to its neighbor j.

Upon sending a task to agent j, agent i immediately gets the reward single $r(s_x, j)$ from agent j. The reward $r(\langle t_x, w_x \rangle, j)$ is the estimated probability that the task x will be allocated based on agent j's both policies for local allocation and task routing:

$$r(s_x, j) = p_j(x) + (1 - p_j(x)) \sum_{k \in \text{neighbors of} j} \pi_{2j}(s'_x, k) * Q_j(s'_x, k)$$

where $p_j(x)$ is the probability that agent j will allocate task x locally, π_{2j} is the task routing policy of agent j, and s'_x is the state where agent j makes a decision for forwarding task x. If the state $s_x = \langle t_x, w_x \rangle$, then $s'_x = \langle t_x, w_x - a_{ij} \rangle$, where a_{ij} is the time cost for transferring a task between agent i and agent j.

The probability $p_j(x)$ depends on agent j's allocation policy π_{1j}:

$$p_j(x) = \sum_{s_t} q(\langle s_c, s_t \rangle | t) \pi_{1j}(\langle s_c, s_t \rangle, t)$$

where t is the type of task x, s_c is the current feature vector of resource availability, $q(\langle s_c, s_t \rangle | t)$ is the probability that agent j is on state $\langle s_c, s_t \rangle$ when it allocates tasks with type t, and π_{1j} is the local allocation policy of agent j. The probability $q(\langle s_c, s_t \rangle | t)$ can be directly estimated during the learning.

	Greedy	FDL	SDL	BDL
Local	Best-first	Learning_1	Best-first	Learning_1
Routing	Random	Random	Learning_2	Learning_2

Table 4.1. Distributed resource allocation approaches

The simple version of Q-learning algorithm is used to update agent i's Q-value function:

$$Q_i(s_x, j) = (1 - \alpha) * Q_i(s_x, j) + \alpha * r(x_s, j)$$

where α is a learning rate (usually 0.5 in our experiments). With modified Q-value function, the FAL algorithm updates the task routing policy π_{2i}.

To speed up the learning, we use an idea, called *backward exploration* [50], of using information about the traversed path for exploration in the reverse direction. When agent i transfer task x to its neighbor j, the message that contains pass x can take along reward information $r(s_x, i)$ of agent i for allocating x. This reward information can be used by agent j to update its own estimate pertaining to i. Later when agent j has to make a decision, it has the updated Q-value for i. As a result, backward exploration speeds up the learning.

4.5 Experiments

4.5.1 Experiment Design

To evaluate the performance of learning models developed above, we compared five resource allocation approaches: *greedy allocation*, *first-decision (local allocation) learning* (FDL), *second-decision (task routing) learning* (SDL), *both-decision learning* (BDL), and *centralized allocation*. The first four approaches are distributed techniques. As shown in Table 4.1, they use different algorithms for each decision-making. The *best-first* algorithm, at each time step, first sorts all received tasks in a descending order of utility rate and then uses the best-fit algorithm in Sharc [113] to allocate tasks one by one. *Learning*$_1$ and

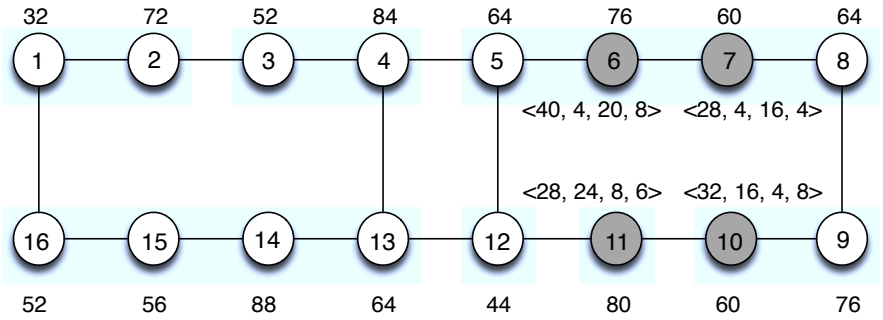


Figure 4.1. The network topology with 16 clusters

Learning$_2$ respectively refer to the learning algorithms we developed for local allocation and task routing. The *random* algorithm for task routing picks a random neighbor to forward an unallocated task. The *centralized allocation* approach has only one manager that fully controls all computing nodes and uses *best-first* algorithm to directly allocate tasks to resources without any routing.

We have tested approaches on several network topologies with 2, 4, 8, and 16 clusters, all of which show similar results. Here we present detailed results for a network topology with 16 clusters pictured in Figure 4.1, each of which uses Sharc to manage its local resources. The number outside a circle represents the number of computing nodes of that cluster. The CPU capacity and network capacity vary on different computing nodes, whose range is in $[50, 150]$.

Our experiments use four types of tasks: *ordinary*, *IO-intense*, *compute-intense*, and *demanding*. Their feature vectors are respectively $\langle 20, 1, 9, 8\rangle$, $\langle 35, 6, 15, 48\rangle$, $\langle 30, 5, 45, 8\rangle$ and $\langle 50, 25, 47, 43\rangle$, each of which shows the mean of service time, utility rate, CPU demand, and network demand. All tasks have waiting time $w = 10$. The service time is under exponential distribution and the rest is under Poisson distribution. Note that the more demanding tasks usually have much higher utility rates.

Only four clusters, shaded in Figure 4.1, receive tasks from external environment. We tested two different task loads: *heavy* and *light*. The vector besides each shaded node

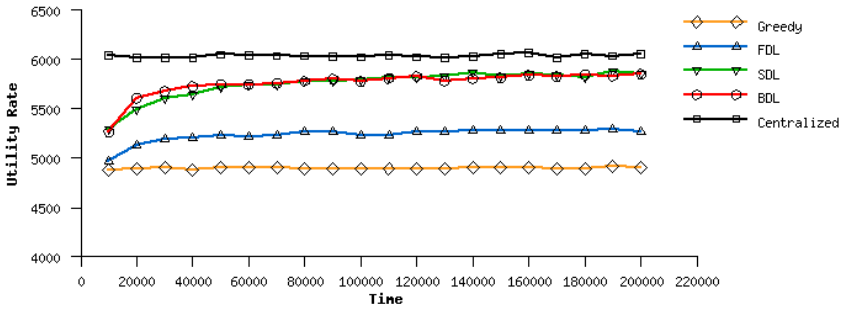


Figure 4.2. Utility rate under light task load

shows, under heavy load, the average number of tasks of four types arriving on that node. Under light task loads, these average numbers are half of those of heavy task loads. Task arrivals of each type on each cluster are under a Poisson distribution. Communication limitation for each cluster is 40 tasks per time step.

In our experiments, availability of each resource is categorized into three levels. All performance measures shown below are computed every 5000 time steps. Results are then averaged over 10 simulation runs and the deviation is computed across the runs.

4.5.2 Results & Discussions

Figure 4.2 shows utility rate trends of the whole cluster network as it runs with different approaches in a lightly loaded environment. The curved lines of FDL, SDL, and BDL demonstrate that local allocation learning, task routing learning and their combination monotonically improve system performance. Under light load where the demand for resources is less than the supply, the best solution is to allocate all received tasks within the system. In such a setting, the centralized allocation approach generates the optimal solution. For distributed allocation approaches, how to route tasks and balance the loads across clusters becomes very important. From Figure 4.2, it can be seen that the performance of SDL and BDL is close to the optimal approach and much better than FDL and the greedy approach. So learning task routing policy works effectively.

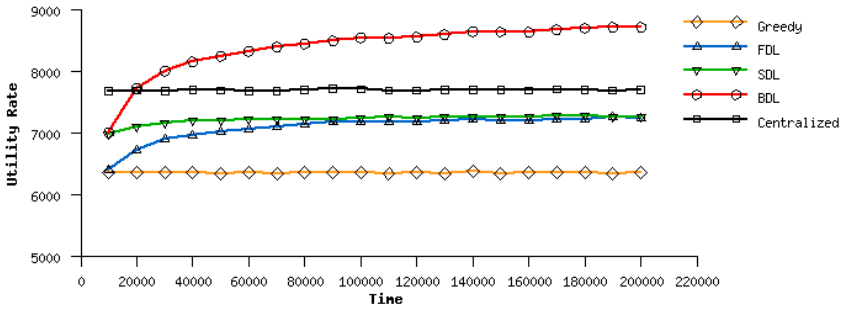


Figure 4.3. Utility rate under heavy task load

When task loads are well-balanced across clusters, resources of each cluster usually can meet tasks' demand and the best-first algorithm is almost optimal for local allocation decisions. In some sense, the similar performance between SDL and BDL verifies the effectiveness of learning local allocation policies. However, when task loads are not well distributed across the clusters, some clusters received more tasks than their capacity. In such a situation, the best-first algorithm will not be optimal, because it does not take into account future task arrival patterns in its current decisions. In contrast, the learning approach implicitly estimates future task arrival patterns and give up some tasks with low utilities and reserve resources for future tasks with high utilities. Therefore, FDL will outperforms the greedy algorithm.

Figure 4.3 show utility rate trends of the cluster network under the heavy load. Most analysis results for the lightly loaded case also holds in the heavily loaded case. In this more complicated case, one significant observation is that BDL outperforms the centralized allocation approach. Under the heavy load, the overall demand for resources exceeds their supply by the whole cluster network. Without considering future task arrivals, the best-first centralized allocation approach is not optimal in such a situation. On the other hand, the learning approach implicitly takes account of future tasks for making current decisions and can work better than the best-first algorithm, which is verified by the performance of FDL and the greedy approach. Combined with effective learned routing policies, the advantage

Approaches	Utility	CPU	Network	Hops
Greedy	4900 ± 28	0.62 ± 0.00	0.60 ± 0.00	1.80 ± 0.01
FDL	5281 ± 41	0.60 ± 0.00	0.58 ± 0.00	3.88 ± 0.04
SDL	5851 ± 37	0.74 ± 0.00	0.71 ± 0.00	1.50 ± 0.02
BDL	5857 ± 39	0.70 ± 0.00	0.67 ± 0.00	4.30 ± 0.06
Centralized	6038 ± 47	0.77 ± 0.00	0.74 ± 0.00	0.00 ± 0.00

Table 4.2. Performance with light load

Approaches	Utility	CPU	Network	Hops
Greedy	6364 ± 30	0.79 ± 0.00	0.76 ± 0.00	2.31 ± 0.01
FDL	7249 ± 29	0.71 ± 0.00	0.69 ± 0.00	4.33 ± 0.06
SDL	7273 ± 27	0.92 ± 0.00	0.89 ± 0.00	2.13 ± 0.01
BDL	8719 ± 49	0.87 ± 0.00	0.85 ± 0.00	5.33 ± 0.05
Centralized	7700 ± 33	0.95 ± 0.00	0.93 ± 0.00	0.00 ± 0.00

Table 4.3. Performance with heavy load

of learning local allocation offsets disadvantages due to partial information and distributed resource control in distributed approaches, which allows BDL to performs better than the centralized allocation approach.

Table 4.2 and 4.3 respectively summarize the performance measures (including utility rates, CPU utilization, network utilization, and task routing hops) of different approaches under light and heavy load during the last 5000 time period of simulations. Under light load, although BDL performs very well in a distributed way, the difference between its utility rate and the optimal one (generated by the centralized approach) is still noticeable, which is around 3%. Several factors contribute to this gap. First, due to partial observation, distributed learned routing policies can not be perfect. In addition, the communication of each agent is limited. As a result, some tasks are not allocated before their deadline. Second, to reduce the policy search space, both learning models use both approximate state space and action space, which introduces further uncertainty that has the effect of decreased performance. We tested more accurate models, such as discretizing availability of each resource into more levels and using more task features in addition to the type task

to represent actions. Although experiment results are slightly better, the learning converges much slower and has poor policies for a long period. Third, the learning never stops its exploration.

Note that BDL has both lower CPU and network utilization than SDL, although it performs better. This is because, with a learned local allocation policy, an agent is willing to give up tasks with low utility and reserve resources for future high-utility tasks, which causes resources to be idle for a higher percentage of the time. This reason also explains that the greedy approach and SDL have less hops per task than both FDL and BDL. The *hops* describes the average number times that a task has been transfered before being allocated. The giving-up behavior causes more tasks to be forwarding in the system, especially under heavy load.

4.6 Related Work

Several distributed scheduling algorithms based on heuristics are developed for allocating tasks with deadlines and resource requirements in [80]. Unlike our approach, both their basis algorithms, *focused address algorithm* and *bidding algorithm*, assume each agent can interact with all other agents and request resource information from them in a real-time manner. As a result, these algorithms have potential scalability issues.

A different resource allocation model is formulated in [83], which assumes a strict separation between agents and resources. Jobs arrive at agents who use reinforcement learning to make decisions about where to execute them and the resources are passive (i.e., do not make decisions) and dedicated. Therefore, there is no direct interaction between agents. The work in [109] has a similar model, but there is a resource arbiter who dynamically decides resource allocation based on value functions of agents, which are learned independently.

Reinforcement learning has been applied to network routing [24, 50]. In their problems, each package has a pre-specified destination, so the routing is targeted. In contrast, in our

problem, agents do not know the destination for an task, which is supposed to be learned. In addition, our task routing learning is also affected by the local allocation learning.

4.7 Summary

In this chapter, we provided a practical application domain for multi-agent learning. The empirical results showed that multi-agent learning was a promising and practical approach to online resource allocation in real computing infrastructures with a network of shared clusters. Compared with a single global learning, multi-agent learning scales up to many applications by partitioning state and action spaces over agents and through concurrent learning over more computational hardware. This work also plausibly suggests that multi-agent learning may be an approach to address online optimization problems in distributed systems, such as large-scale grid computing, sensor networks, and peer-to-peer information retrieval.

Through this application work, we also observe limitations of this MARL approach. This approach converges slowly with even 16 agents and does not scale well as the number of agents increases, which motivates our research to develop a supervision framework to coordinate and improve MARL (which will be presented in Part III). In addition, although the FAL algorithm performs effectively in this application, it may not converge in competitive scenarios. As shown in Figure 1.2, our MARL paradigm tends to design local reward signals (instead of using the single global reward) for agents's learning, which are more efficiently computed and specifically tailored to their behaviors. By using local reward signals, competitive learning scenarios may occur event in cooperative MAS. In the next chapter, we will describe our MARL algorithms that consider both competitive and non-competitive multi-agent settings.

CHAPTER 5

MULTI-AGENT LEARNING WITH POLICY PREDICTION

In our previous chapter, we design a gradient-based multi-agent reinforcement learning (MARL) algorithm, called Fair Action Learner (FAL), and apply it to a complex MAS application. Although FAL performs effectively in that application, it may not converge in competitive scenarios. As shown in Figure 1.2, our MARL paradigm tends to design local reward signals (instead of using the single global reward) for agents's learning in large complex applications, which are more efficiently computed and specifically tailored to their behaviors. By using local reward signals, competitive learning scenarios may occur event in cooperative MAS. In the next chapter, we will present our MARL algorithms that consider both competitive and non-competitive multi-agent settings. The work of this chapter was published in AAAI 2010 [124].

Several MARL algorithms have been proposed and studied [92, 23, 40, 22, 28, 11], all of which have some theoretical results of convergence in general-sum games. A common assumption of these algorithms is that an agent (or player) knows its own payoff matrix. To guarantee convergence, each algorithm has its own additional assumptions, such as requiring an agent to know a Nash Equilibrium (NE) and the strategy of the other players[23, 11, 28], or observe what actions other agents executed and what rewards they received [40, 28]. For practical applications, these assumptions are very constraining and unlikely to hold, and, instead, an agent can only observe the immediate reward after selecting and performing an action.

In this chapter, we first propose a new gradient-based algorithm that uses policy prediction in a basic gradient ascent algorithm. The key idea behind this algorithm is that a player

adjusts its strategy in response to forecasted strategies of the other players, instead of their current ones. We analyze this algorithm in two-person, two-action, general-sum iterated game and prove that if at least one player uses this algorithm (if not both, assume the other player uses the standard gradient ascent algorithm), then players' strategies will converge to a Nash equilibrium. Like other MARL algorithms, besides the common assumption, this algorithm also has additional requirements that a player knows the other player's strategy and current strategy gradient (or payoff matrix) so that it can forecast the other player's strategy.

Motivated by our theoretical convergence analysis, we then propose a new practical MARL algorithm exploiting the idea of policy prediction. Our practical algorithm only requires an agent to observe its reward of choosing a given action. We show that our practical algorithm can learn an optimal policy when other players use stationary policies. Empirical results show that it converges in more situations than that covered by our formal analysis. Compared to state-of-the-art MARL algorithms, WPL [3], WoLF-PHC [23] and GIGA-WoLF [22], it empirically converges faster and in a wider variety of situations.

In the remainder of this chapter, we first review the basic gradient ascent algorithm and then introduce our gradient-based algorithm followed by its theoretical analysis. We then describe a new practical MARL algorithm and evaluate it in benchmark games, distributed task allocation problem and network routing. Finally, this chapter is summarized.

5.1 Gradient Ascent

In this section, we will first define some notations that will be used in this chapter. We then present a brief overview of normal-form games and review the basic gradient ascent algorithm.

5.1.1 Notation

- Δ denotes the valid strategy space, i.e., [0, 1].

- $\Pi_\Delta : \Re \rightarrow \Delta$ denotes the projection to the valid space,

$$\Pi_\Delta[x] = \text{argmin}_{z \in \Delta} |x - z|.$$

- $P_\Delta(x, v)$ denotes the projection of a vector v on $x \in \Delta$,

$$P_\Delta(x, v) = \lim_{\eta \rightarrow 0} \frac{\Pi_\Delta(x + \eta v) - x}{\eta}$$

5.1.2 Normal-Form Games

A two-player, two-action, general-sum normal-form game is defined by a pair of matrices

$$R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \text{ and } C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

specifying the payoffs for the *row* player and the *column* player, respectively. The players simultaneously select an action from their available set, and the joint action of the players determines their payoffs according to their payoff matrices. If the row player and the column player select action $i, j \in \{1, 2\}$, respectively, then the row player receives a payoff r_{ij} and the column player receives the payoff c_{ij}.

The players can choose actions stochastically based on some probability distribution over their available actions. This distribution is said to be a mixed strategy. Let $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ denote the probability of choosing the first action by the row and column player, respectively. With a joint strategy (α, β), the row player's expected payoff is

$$\begin{aligned} V_r(\alpha, \beta) &= r_{11}(\alpha\beta) + r_{12}(\alpha(1 - \beta)) + r_{21}((1 - \alpha)\beta) \\ &\quad + r_{22}((1 - \alpha)(1 - \beta)) \end{aligned} \tag{5.1}$$

and the column player's expected payoff is

$$\begin{aligned} V_c(\alpha,\beta) &= c_{11}(\alpha\beta) + c_{12}(\alpha(1-\beta)) + c_{21}((1-\alpha)\beta) \\ &\quad + c_{22}((1-\alpha)(1-\beta)). \end{aligned} \tag{5.2}$$

A joint strategy (α^*, β^*) is said to be a *Nash equilibrium* if (i) for any mixed strategy α of the row player, $V_r(\alpha^*, \beta^*) \geq V_r(\alpha, \beta^*)$, and (ii) for any mixed strategy β of the column player, $V_c(\alpha^*, \beta^*) \geq V_c(\alpha^*, \beta)$. In other words, no player can increase its expected payoff by changing its equilibrium strategy unilaterally. It is well-known that every game has at least one Nash equilibrium.

5.1.3 Learning using Gradient Ascent in Iterated Games

In an iterated normal-form game, players repeatedly play the same game. Each player seeks to maximize it own expected payoff in response to the strategy of the other player. Using the gradient ascent algorithm, a player can increase its expected payoff by moving its strategy in the direction of the current gradient with some step size. The gradient is computed as the partial derivative of the agent's expected payoff with respect to its strategy:

$$\begin{aligned} \partial_\alpha V_r(\alpha,\beta) &= \frac{\partial V_r(\alpha,\beta)}{\partial \alpha} = u_r\beta + b_r \\ \partial_\beta V_c(\alpha,\beta) &= \frac{\partial V_c(\alpha,\beta)}{\partial \beta} = u_c\alpha + b_c \end{aligned} \tag{5.3}$$

where $u_r = r_{11} + r_{22} - r_{12} - r_{21}, b_r = r_{12} - r_{22}, u_c = c_{11} + c_{22} - c_{12} - c_{21}$, and $b_c = c_{21} - c_{22}$.

If (α_k, β_k) are the strategies on the kth iteration and both players use gradient ascent, then the new strategies will be:

$$\begin{aligned} \alpha_{k+1} &= \Pi_\Delta[\alpha_k + \eta\partial_\alpha V_r(\alpha_k,\beta_k)] \\ \beta_{k+1} &= \Pi_\Delta[\beta_k + \eta\partial_\beta V_c(\alpha_k,\beta_k)] \end{aligned} \tag{5.4}$$

where η is the gradient step size. If the gradient moves the strategy out of the valid probability space, then the function Π_Δ will project it back. This will only occur on the boundaries (i.e., 0 and 1) of the probability space.

Singh, Kearns, and Mansour (2000) analyzed the gradient ascent algorithm by examining the dynamics of the strategies in the case of an infinitesimal step size ($\lim_{\eta \to 0}$). This algorithm is called *Infinitesimal Gradient Ascent (IGA)*. Its main conclusion is that, if both players use IGA, their average payoffs will converge in the limit to the expected payoffs for some Nash equilibrium.

Note that the convergence result of IGA focuses on the average payoffs of the two players. This notion of convergence is still weak, because, although the players' average payoffs converge, their strategies may not converge to a Nash equilibrium (e.g., in zero-sum games). In the next section, we will describe a new gradient ascent algorithm with policy prediction that allows players' strategies to converge to a Nash equilibrium.

5.2 Gradient Ascent With Policy Prediction

As shown in Equation 5.4, the gradient used by IGA to adjust the strategy is based on current strategies. Suppose that one player knows its change direction of the opponent's strategy, i.e., strategy derivative, in addition to its current strategy. Then the player can forecast the opponent's strategy and adjust its strategy in response to the forecasted strategy. Thus the strategy update rules is changed to:

$$\begin{aligned} \alpha_{k+1} &= \Pi_\Delta[\alpha_k + \eta \partial_\alpha V_r(\alpha_k, \beta_k + \gamma \partial_\beta V_c(\alpha_k, \beta_k))] \\ \beta_{k+1} &= \Pi_\Delta[\beta_k + \eta \partial_\beta V_c(\alpha_k + \gamma \partial_\alpha V_r(\alpha_k, \beta_k), \beta_k)] \end{aligned} \tag{5.5}$$

The new derivative terms with γ serve as a short-term prediction (i.e., with length γ) of the opponent's strategy. Each player computes its strategy gradient based on the forecasted strategy of the opponent. If the prediction length $\gamma = 0$, the algorithm is actually IGA.

Because of using policy prediction (i.e., $\gamma > 0$), we call this algorithm IGA-PP (for theoretical analysis, we also consider the case of an infinitesimal step size ($\lim_{\eta \to 0}$)). As will be shown in the next section, if one player uses IGA-PP and the other uses IGA-PP or IGA, their strategies will converge to a Nash equilibrium.

The prediction length γ will affect the convergence of the IGA-PP algorithm. With a too large prediction length, a player may not predict the opponent strategy in a right way. Then the gradient based on the wrong opponent strategy deviates too much from the gradient based on the current strategy, and the player adjusts its strategy in a wrong direction. As a result, in some cases (e.g., $u_r u_c > 0$), players' strategies converge to a point that is not a Nash equilibrium. The following conditions restrict γ to be appropriate.

Condition 1: $\gamma > 0$,

Condition 2: $\gamma^2 u_r u_c \neq 1$,

Condition 3: for any $x \in \{b_r, u_r + b_r\}$ and $y \in \{b_c, u_c + b_c\}$, if $x \neq 0$, then $x(x + \gamma u_r y) > 0$, and if $y \neq 0$, then $y(y + \gamma u_c x) > 0$.

Condition 3 basically says the term with γ will not change the sign of the x or y, and a sufficiently small $\gamma > 0$ will always satisfy them.

5.3 Analysis of IGA-PP

In this section, we will prove the following main result.

Theorem 1. *If, in a two-person, two-action, iterated general-sum game, both players follow the IGA-PP algorithm (with sufficiently small $\gamma > 0$), then their strategies will asymptotically converge to a Nash equilibrium.*

Similar to the analysis in [92, 23], our proof of this theorem is accomplished by examining the possible cases of the dynamics of players' strategies following IGA-PP, as done

by Lemma 3, 4, and 5. To facilitate the proof, we first prove that if players' strategies converge by following IGA-PP, then they must converge to a Nash equilibrium, i.e., Lemma 2. Note that our proof assumes that each player learns and acts based on expected rewards, instead of the immediate rewards, which results in deterministic processes. This assumption is different from reinforcement learning (i.e., Q-learning) that uses immediate rewards and results in stochastic processes.

For brevity, let ∂_{α_k} denote $\partial_\alpha V_r(\alpha_k, \beta_k)$, and ∂_{β_k} denote $\partial_\beta V_c(\alpha_k+, \beta_k)$. We reformulate the update rules of IGA-PP from Equation 5.5 using Equation 5.3:

$$\begin{aligned} \alpha_{k+1} &= \Pi_\Delta[\alpha_k + \eta(\partial_{\alpha_k} + \gamma u_r \partial_{\beta_k})] \\ \beta_{k+1} &= \Pi_\Delta[\beta_k + \eta(\partial_{\beta_k} + \gamma u_c \partial_{\alpha_k})] \end{aligned} \tag{5.6}$$

Lemma 1. *If the projected partial derivatives at a strategy pair (α^*, β^*) are zero, that is, $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$, then (α^*, β^*) is a Nash equilibrium.*

Proof. Assume that (α^*, β^*) is not a Nash equilibrium. Then at least one player, say the column player, can increase its expected payoff by changing its strategy unilaterally. Let the improved point be (α^*, β). Because the strategy space Δ is convex and the linear dependence of $V_c(\alpha, \beta)$ on β, then, for any $\epsilon > 0$, $(\alpha^*, (1-\epsilon)\beta^* + \epsilon\beta)$ must also be an improved point, which implies the projected gradient of β at (α^*, β^*) is not zero. By contradiction, (α^*, β^*) is a Nash equilibrium. □

Lemma 2. *If, in following IGA-PP with sufficiently small $\gamma > 0$, $\lim_{k\to\infty}(\alpha_k, \beta_k) = (\alpha^*, \beta^*)$, then (α^*, β^*) is a Nash equilibrium.*

Proof. The strategy pair trajectory converges at (α^*, β^*) if and only if the projected gradients used by IGA-PP are zero, that is, $P_\Delta(\alpha^*, \partial_{\alpha^*} + \gamma u_r \partial_{\beta^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*} + \gamma u_c \partial_{\alpha^*}) = 0$. Now we are showing that $P_\Delta(\alpha^*, \partial_{\alpha^*} + \gamma u_r \partial_{\beta^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*} + \gamma u_c \partial_{\alpha^*}) = 0$ will imply $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$, which, according to

Lemma 1, will finish the proof and indicates (α^*, β^*) is a Nash equilibrium. Assume $\gamma > 0$ is sufficiently small that satisfies Condition 2 and 3. Consider three possible cases when the projected gradients used by IGA-PP are zero.

Case 1: both gradients are zero, that is, $\partial_{\alpha^*} + \gamma u_r \partial_{\beta^*} = 0$ and $\partial_{\beta^*} + \gamma u_c \partial_{\alpha^*} = 0$. By solving them, we get $(1 - \gamma^2 u_r u_c)\partial_{\alpha^*} = 0$ and $\partial_{\beta^*} = -\gamma u_c \partial_{\alpha^*}$, which implies $\partial_{\alpha^*} = 0$ and $\partial_{\beta^*} = 0$, due to Condition 2 (i.e., $\gamma^2 u_r u_c \neq 1$). Therefore, $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$.

Case 2: at least one gradient is greater than zero. Without loss of generality, assume $\partial_{\alpha^*} + \gamma u_r \partial_{\beta^*} > 0$. Because its projected gradient is zero, its strategy is on the boundary of the strategy space Δ, which implies $\alpha^* = 1$. Now we consider three possible cases of the column player's partial strategy derivative $\partial_{\beta^*} = u_c \alpha^* + b_c = u_c + b_c$.

1. $\partial_{\beta^*} = 0$, which implies $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$. $\partial_{\alpha^*} + \gamma u_r \partial_{\beta^*} > 0$ and $\alpha^* = 1$ implies $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$.

2. $\partial_{\beta^*} = u_c + b_c > 0$, due to Condition 3, implies $\partial_{\beta^*} + \gamma u_c \partial_{\alpha^*} > 0$. Because the projected gradient of β^* is zero, then $\beta^* = 1$, which implies $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$. $\partial_{\alpha^*} + \gamma u_r \partial_{\beta^*} = u_r + b_r + \gamma u_r (u_c + b_c) > 0$ and Condition 3 implies $\partial_{\alpha^*} = u_r + b_r > 0$, which, combined with $\alpha^* = 1$, implies $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$.

3. $\partial_{\beta^*} = u_c + b_c < 0$. The analysis of this case is similar to the case above with $\partial_{\beta^*} > 0$, except $\beta^* = 0$.

Case 3: at least one gradient is less than zero. The proof of this case is similar to Case 2. Without loss of generality, assume $\partial_{\alpha^*} + \gamma u_r \partial_{\beta^*} < 0$, which implies $\alpha^* = 0$. Then using Condition 3 and analyzing three cases of $\partial_{\beta^*} = u_c \alpha^* + b_c = b_c$ will also get $P_\Delta(\alpha^*, \partial_{\alpha^*}) = 0$ and $P_\Delta(\beta^*, \partial_{\beta^*}) = 0$.

□

To prove IGA-PP's Nash convergence, we now will examine the dynamics of the strategy pair following IGA-PP. The strategy pair (α, β) can be viewed as a point in $\Re^2$ constrained to lie in the unit square. Using Equation 5.3, 5.6, and an infinitesimal step size, it is easy to show that the *unconstrained* dynamics of the strategy pair is defined by the following differential equation

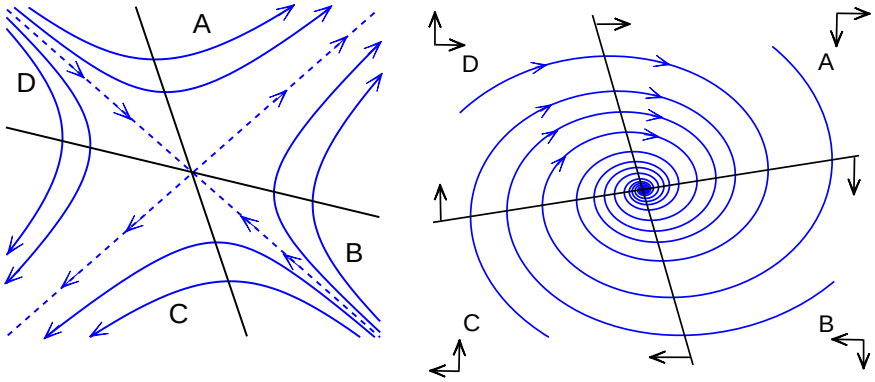
$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \gamma u_r u_c & u_r \\ u_c & \gamma u_c u_r \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \begin{bmatrix} \gamma u_r b_c + b_r \\ \gamma u_c b_r + b_c \end{bmatrix} \tag{5.7}$$

We denote the 2×2 matrix in Equation 5.7 as U.

In the unconstrained dynamics, there exists at most one point of zero-gradient, which is called the center (or origin) and denoted (α^c, β^c). If the matrix U is invertible, by setting the left hand side of Equation 5.7 to zero, using Condition 2 (i.e., $\gamma^2 u_r u_c < 1$), and solving for the center, we get

$$(\alpha^c, \beta^c) = (\frac{-b_r}{u_r}, \frac{-b_c}{u_c}). \tag{5.8}$$

Note that the center is in general not at (0, 0) and may not even be in the unit square.



a) A saddle at the center b) A stable focus at the center

Figure 5.1. The phase portraits of the IGA-PP dynamics: a) when U has real eigenvalues and b) when U has imaginary eigenvalues with negative real part

From dynamical systems theory [76], if the matrix U is invertible, qualitative forms of the dynamical system specified by Equation 5.7 depend on eigenvalues of U, which are given by

$$\lambda_1 = \gamma u_r u_c + \sqrt{u_r u_c} \text{ and } \lambda_2 = \gamma u_r u_c - \sqrt{u_r u_c}. \tag{5.9}$$

If U is invertible, $u_r u_c \neq 0$. If $u_r u_c > 0$, then U has two real eigenvalues; otherwise, U has two imaginary conjugate eigenvalues with negative real part (because $\gamma > 0$). Therefore, based on linear dynamical systems theory, if U is invertible, Equation 5.7 has two possible phase portraits shown in Figure 5.1. In each diagram, there are two axes across the center. Each axis is corresponding to one player, whose strategy gradient on this axis are zero. Because $u_r, u_c \neq 0$ in Equation 5.7, two axes are off the horizonal or the vertical line and not orthogonal to each other. These two axes produce four quadrants.

To prove Theorem 1, we only need to show that IGA-PP always leads the strategy pair to converge a Nash equilibrium in three mutually exclusive and exhaustive cases:

- $u_r u_c = 0$, i.e., U is not invertible,
- $u_r u_c < 0$, i.e., having a *saddle* at the center,
- $u_r u_c > 0$, i.e., having a *stable focus* at the center.

Lemma 3. *If U is not invertible, for any initial strategy pair, IGA-PP (with sufficiently small γ) leads the strategy pair trajectory to converge to a Nash equilibrium.*

Proof. If U is not invertible, $\det(U) = (\gamma^2 u_r u_c - 1) u_r u_c = 0$. A sufficiently small γ will always satisfy Condition 2, i.e., $\gamma^2 u_r u_c \neq 1$. Therefore, u_r or u_c is zero. Without loss of generality, assume u_r is zero. Then the gradient for the row player is constant (See Equation 5.7), i.e., b_r. As a result, if $b_r = 0$, then its strategy α keeps on its initial value; otherwise, its strategy will converge to $\alpha = 0$ (if $b_r < 0$) or $\alpha = 1$ (if $b_r > 0$). After the row player's strategy α becomes a constant, due to $u_r = 0$, the column player's strategy gradient also becomes a constant. Then its strategy β stays on a value (if the gradient is zero) or converges to one or zero, depending on the sign of the gradient. According to Lemma 2, the joint strategy converges to a Nash equilibrium. □

Lemma 4. *If U has real eigenvalues, for any initial strategy pair, IGA-PP leads the strategy pair trajectory to converge to a point on the boundary that is a Nash equilibrium.*

Proof. From Equation 5.9, real eigenvalues implies $u_r u_c > 0$. Assume $u_r > 0$ and $u_c > 0$ (the analysis for the case with $u_r < 0$ and $u_c < 0$ is analogous and omitted). In this case, the dynamics of the strategy pair has the qualitative form shown in Figure 5.1a.

Consider the case when the center is inside the unit square. If the initial point is at the center where the gradient is zero, it converges immediately. For an initial point in quadrant B or D, if it is on the dashed line, the trajectory will asymptotically converge to the center; otherwise, the trajectory will eventually enter either quadrant A or C. Any trajectory in quadrant A (or C) will converge to the top-right corner (or the bottom-left corner) of the unit square. Therefore, by Lemma 2, any trajectory always converges a Nash equilibrium. Cases when the center on the boundary or outside the unit square can be shown similarly to converge, and are discussed in [92]. □

Lemma 5. *If U has two imaginary conjugate eigenvalues with negative real part, for any initial strategy pair, the IGA-PP algorithm leads the strategy pair trajectory to asymptotically converge to a point that is a Nash equilibrium.*

Proof. From dynamical systems theory [76], if U has two imaginary conjugate eigenvalues with negative real part, the unconstrained dynamics of Equation 5.7 has a stable focus at the center, which means, starting from any point, the trajectory will asymptotically converge to the center (α^c, β^c) in a spiral way. From Equation 5.9, the imaginary eigenvalues implies $u_r u_c < 0$. Assume $u_r > 0$ and $u_c < 0$ (the case with $u_r < 0$ and $u_c > 0$ is analogous), whose general phase portrait is shown in Figure 5.1b. One observation is that the direction of the gradient of the strategy pair changes in a clockwise way through the quadrants.

By Lemma 2, we only need to show the strategy pair trajectory will converge a point in the constrained dynamics. We analyze three possible cases to consider depending on the location of the center (α^c, β^c).

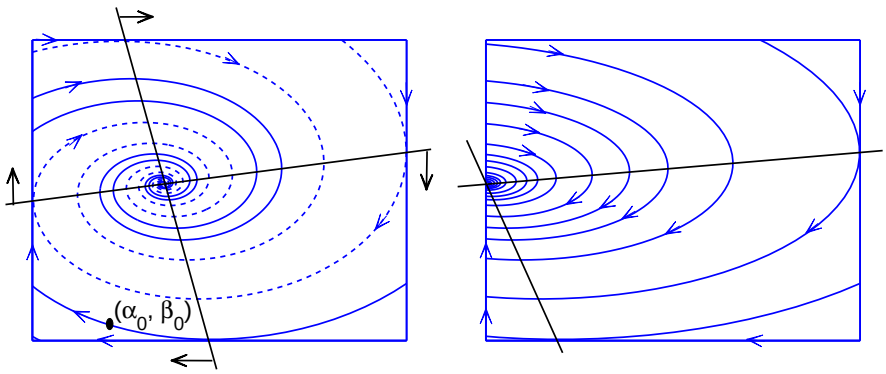


Figure 5.2. Example dynamics when U has imaginary eigenvalues with negative real part

1. **Center in the interior of the unit square.** First, we observe that all boundaries of the unit square are tangent to some spiral trajectory, and at least one boundary is tangent to a spiral trajectory, whose remaining part after the tangent point lies entirely within the unit square, e.g., two dashed trajectories in Figure 5.2a.

   If the initial strategy pair coincidentally is the center, it will always stay because its gradient is zero. Otherwise, the trajectory starting from the initial point either does not intersect any boundary, which will asymptotically converge to the center, or intersects with a boundary. In the latter case, when the trajectory hits a boundary, it then travels along the boundary until it reaches the point at which the boundary is tangent to some spiral, whose remaining part after the tangent point may or may not lie entirely within the unit square. If it does, then the trajectory will converge to the center along that spiral. If it does not, the trajectory will follow the tangent spiral to the next boundary in the clockwise direction. This process repeats until the boundary is reached that is tangent to a spiral, whose remaining part after the tangent point lies entirely within the unit square. Therefore, the trajectory will eventually asymptotically converge to the center.

2. **Center on the boundary.** Consider the case where the center is on the left-side boundary of the unit square, as shown in Figure 5.2b. For convenience, assume the top left corner only belongs to the left boundary and the bottom left corner only belongs to the bottom boundary. If the initial strategy pair coincidentally is the center, it will always stay because of its gradient is zero. Otherwise, because of clockwise directions of the gradient, no matter where the trajectory starts, it will always finally hit the left boundary below the center, and then travels up along the left boundary and asymptotically converge to the center. A similar argument can be constructed when the center is on some other boundary of the unit square.

3. **Center outside the unit square.** In this case, the strategy trajectory will converge to some corner of the unit square depending on the location of the unit square, as discussed in [92].

□

Theorem 2. *If, in a two-person, two-action, iterated general-sum game, one player uses IGA-PP (with a sufficiently small γ) and the other player uses IGA, then their strategies will converge to a Nash equilibrium.*

The proof of this theorem is omitted, which is similar to that of Theorem 1.

5.4 A Practical Algorithm

Based on the idea of IGA-PP, we now present a new practical MARL algorithm, called Policy Gradient Ascent with approximate policy prediction (PGA-APP), shown in Algorithm 4. The PGA-APP algorithm only requires the observation of the reward of the selected action. To drop the assumptions of IGA-PP, PGA-APP needs to address the key question: how can an agent estimate its policy gradient with respect to the opponent's forecasted strategy without knowing the current strategy and the gradient of the opponent?

For clarity, let us consider the policy update rule of IGA-PP for the row player, shown by Equation 5.6. IGA-PP's policy gradient of the row player (i.e., $\partial_{\alpha_k} + \gamma u_r \partial_{\beta_k}$) contains two components: its own partial derivative (i.e., ∂_{α_k}) and the product of a constant and the column player's partial derivative (i.e., $\gamma u_r \partial_{\beta_k}$) with respect to the current joint strategies. PGA-APP estimates these two components, respectively.

To estimate the partial derivative with respect to the current strategies, PGA-APP uses Q-learning to learn the expected value of each action in each state. The value function $Q(s, a)$ returns the expected reward (or payoff) of executing action a in state s. The policy $\pi(s, a)$ returns the probability of taking action a in state s. As shown by Line 5 in Algorithm 4, Q-learning only uses the immediate reward to update the expected value. With the value function Q and the current policy π, PGA-APP then can calculate the partial derivative, as shown by Line 8. To illustrate that the calculation works properly, let us consider a two-person, two-action repeated game, where each agent has a single state. Let $\alpha = \pi_r(s, 1)$ and $\beta = \pi_c(s, 1)$ be the probability of the first action of the row player and the column player, respectively. Then $Q_r(s, 1)$ is the expected value of the row player playing the first action, which will converge to $\beta * r_{11} + (1 - \beta) * r_{12}$ by using Q-learning. It is easy to show that, when Q-learning converges, $(Q_r(s, 1) - V(s))/(1 - \pi_r(s, 1)) = u_r\beta + b_r = \frac{\partial V_r(\alpha,\beta)}{\partial \alpha}$, which is the partial derivative of the row player (as shown by Equation 5.3).

Using Equation 5.3, we can expand the second component, $\gamma u_r \partial_{\beta_k} = \gamma u_r u_c \alpha + \gamma u_r b_c$. So it is actually a linear function of the row player's own strategy. PGA-APP approximates the second component by the term $-\gamma|\delta(s, a)|\pi(s, a)$, as shown in Line 9. This approximation has two advantages. First, when players' strategies converge to a Nash equilibrium, this approximated derivative will be zero and will not cause them to deviate from the equilibrium. Second, the negative sign of this approximation term is intended to simulate the partial derivative well for the case with $u_r u_c < 0$ (where IGA does not converge) and allows the algorithm to converge in all cases (properly small γ will allow convergence in other cases, i.e., $u_r u_c \geq 0$). Line 12 projects the adjusted policy to the valid space.

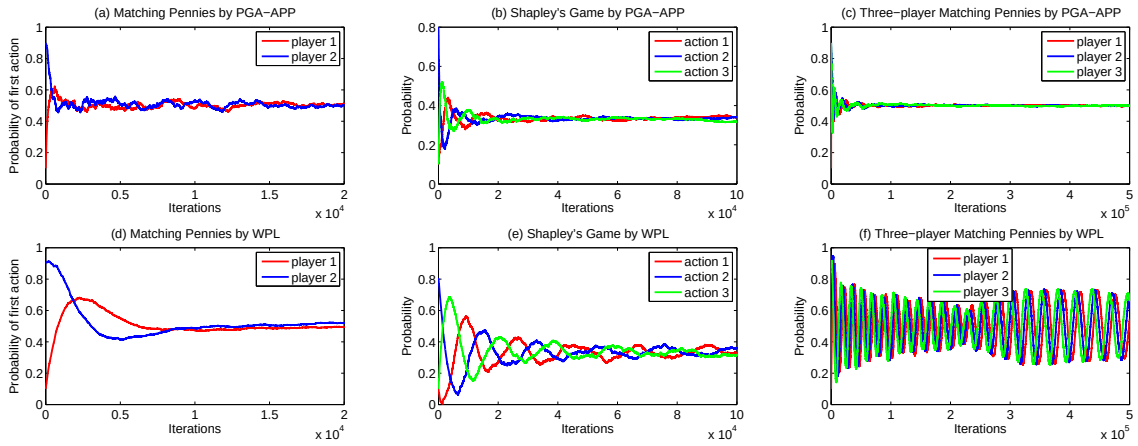


Figure 5.3. Convergence of PGA-APP (on the top row) and WPL (on the bottom row) in games. Plot (a), (c), (d) and (f) shows the dynamics of the probability of the first action of each player, and plot (b) and (e) shows the dynamics of the probability of each action of the first player. Parameters: $\theta = 0.8$, $\xi = 0$, $\gamma = 3$, $\eta = 5/(5000 + t)$ for PGA-APP (η is tuned and decayed slower for WPL), where t is the current number of iterations, and a fixed exploration rate = 0.05. Value function Q is initialized with zero. For two-action games, players' initial policies are (0.1, 0.9) or (0.9, 0.1), respectively, and, for three-action games, their initial policies are (0.1, 0.8, 0.1) and (0.8, 0.1, 0.1).

In some sense, PGA-APP extends Q-learning and is capable of learning mixed strategies. A player following PGA-APP with $\gamma < 1$ will learn an optimal policy if the other players are playing stationary policies. It is because, with a stationary environment, using Q-learning, the value function Q will converge to the optimal one, denoted by Q^*, with a suitable exploration strategy. With $\gamma < 1$, the approximate derivative term in Line 9 will never change the sign of the gradient, and policy π converges to a policy that is greedy with respect to Q. So when Q is converging to Q^*, π converges to a best response.

Learning parameters will affect the convergence of PGA-APP. For competitive games (with $u_r u_c < 0$), the larger the derivative prediction length γ, the faster the convergence. But for non-competitive games (with $u_r u_c \geq 0$), too large γ will violate Condition 3 and cause players' strategies to converge to a point that is not a Nash equilibrium. With higher learning rates θ and η, PGA-APP learns a policy faster at the early stage but the policy may oscillate at late stages. Properly decaying θ and η makes PGA-APP converge better.

Algorithm 4: PGA-APP Algorithm

Let θ and η be the learning rates, ξ be the discount factor, γ be the derivative prediction length;
Initialize value function Q and policy π;
repeat
  Select an action a in current state s according to policy $\pi(s, a)$ with suitable exploration ;
  Observing reward r and next state s', update $Q(s, a) \leftarrow (1 - \theta)Q(s, a) + \theta(r + \xi \max_{a'} Q(s', a'))$;
  Average reward $V(s) \leftarrow \sum_{a \in A} \pi(s, a)Q(s, a)$;
  foreach *action* $a \in A$ **do**
    if $\pi(s, a) = 1$ **then** $\hat{\delta}(s, a) \leftarrow Q(s, a) - V(s)$ **else** $\hat{\delta}(s, a) \leftarrow (Q(s, a) - V(s))/(1 - \pi(s, a))$ **;**
    $\delta(s, a) \leftarrow \hat{\delta}(s, a) - \gamma|\hat{\delta}(s, a)|\pi(s, a)$ **;**
    $\pi(s, a) \leftarrow \pi(s, a) + \eta\delta(s, a)$ **;**
  end
  $\pi(s) \leftarrow \Pi_{\Delta}[\pi(s)]$;
until *the process is terminated* ;

However, the initial value and the decay of learning rate η need to be set appropriately for the value of the learning rate θ, because we do not want to take larger policy update steps than steps with which values are updated.

5.4.1 Experiments: Normal-Form Games

(a) Matching Pennies

$$R = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

(b) Shapley's Game

$$R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

(c) Three-Player Matching Pennies

Player 1 gets one dollar for matching player 2;
player 2 gets one dollar for matching player 3;
player 3 gets one dollar for not matching player 1;
otherwise, players get zero.

Figure 5.4. Normal-form games.

We have evaluated PGA-APP, WoLF-PHC [23], GIGA-WoLF [22], and WPL [3] on a variety of normal-form games. Due to space limitation, we only show results of PGA-APP and WPL in three representative benchmark games: matching pennies, Shapley's game,

and three-player matching pennies, as defined in Figure 5.4. The results of WoLF-PHC and GIGA-WoLF have been shown and discussed in [22, 3]. As shown in Figure 5.3, using PGA-APP, players' strategies converge to a Nash equilibrium in all cases, including games with three players or three actions that are not covered by our formal analysis. Therefore, PGA-APP empirically has a stronger convergence property than WPL, WoLF-PHC and GIGA-WoLF, each of which does not converge in one of two games: Shapley's game and three-player matching pennies. Through experimenting with various parameter settings, we also observe that PGA-APP generally converges faster than WPL, WoLF-PHC and GIGA-WoLF. One possible reason is that, as shown in Figure 5.1b, the strategy trajectory following IGA-PP spirals directly into the center, while the trajectory following IGA-WoLF moves along an elliptical orbit in each quadrant and slowly approaches to the center, as discussed in [23].

5.4.2 Experiments: Distributed Task Allocation

We used our own implementation of the distributed task allocation problem (DTAP) that was described in [3]. Agents are organized in a network. Each agent may receive tasks from either the environment or its neighbors. At each time unit, an agent makes a decision for each task received during this time unit whether to execute the task locally or send it to a neighbor for processing. A task to be executed locally will be added to the local first-come-first-serve queue. The main goal of DTAP is to minimize the average total service time (ATST) of all tasks, including routing time, queuing time, and execution time.

We applied WPL, GIGA-WoLF, and PGA-APP, respectively, to learn the policy of deciding where to send a task: the local queue or one of its neighbors. The agent's state is defined by the size of the local queue, which is different from the experiments in [3] (where each agents has a single state). All algorithms use value-learning rate $\theta = 1$ and policy-learning rate $\eta = 0.0001$. PGA-APP used prediction length $\gamma = 1$.

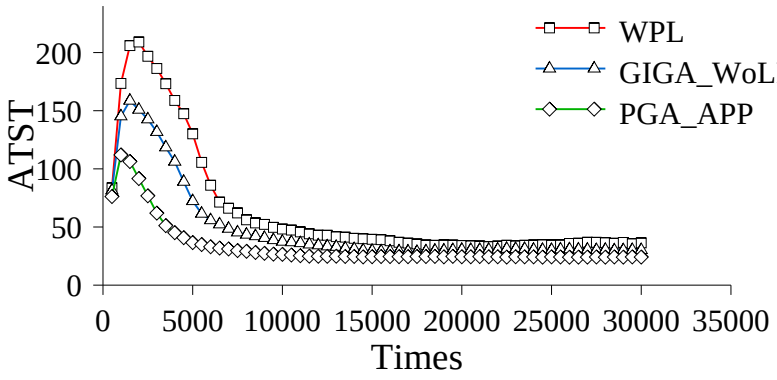


Figure 5.5. Performance in distributed task allocation

Experiments were conducted using uniform two-dimension grid networks of agents with different sizes: 6x6, 10x10, and 18x18, and with different task arrival patterns, all of which show similar comparison results. For brevity, we only present here the results for the 10x10 grid (with 100 agents), where tasks arrive at the 4x4 sub-grid at the center at an average rate 0.5 tasks/time unit. Communication delay between two adjacent agents is one time unit. All agents can execute a task at a rate of 0.1 task/time unit.

Figure 5.5 shows the results of these three algorithms, all of which converge. PGA-APP converges faster and to a better ATST: WPL converges to 34.25 ± 1.46 and GIGA-WoLF to 30.30 ± 1.64, while PGA-APP converges to 24.89 ± 0.82 (results are averaged over 20 runs).

5.4.3 Experiments: Network Routing

We also evaluated PGA-APP in network routing. A network consists of a set of agents and links between them. Packets are periodically introduced into the network under a Poisson distribution with a random origin and destination. When a packet arrives at an agent, the agent puts it into the local queue. At each time step, an agent makes its routing decision of choosing which neighbor to forward the top packet in the queue. Once a packet

reaches its destination, it is removed from the network. The main goal in this problem is to minimize the Average Delivery Time (ADT) of all packets.

We used the experimental setting that was described in [123]. The network is a 10x10 irregular grid with some removed edges. The time cost of sending a packet down a link is a unit cost. The packet arrival rate to the network is 4. Each agent uses the learning algorithm to learn its routing policy.

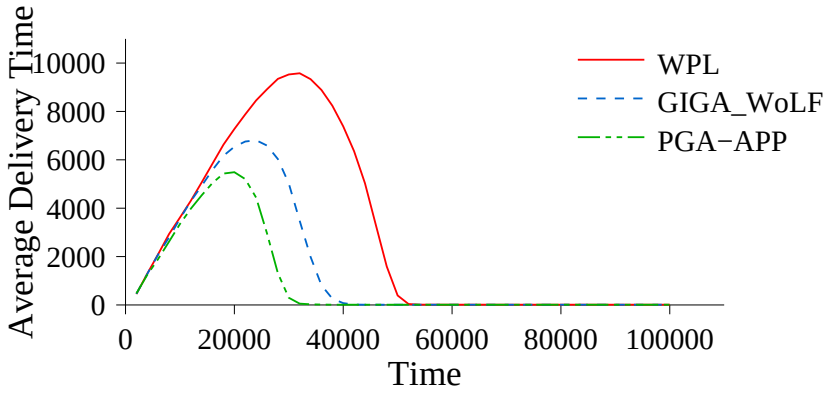


Figure 5.6. Performance in network routing

Figure 5.6 shows the results of applying WPL, GIGA-WoLF, and PGA-APP to this problem. All three algorithms demonstrate convergence, but PGA-APP converges faster and to a better ADT: WPL converges to 11.60 ± 0.29 and GIGA-WoLF to 10.22 ± 0.24, while PGA-APP converges to 9.86 ± 0.29 (results are averaged over 20 runs).

5.5 Summary

The co-adaption of multiple learners in multi-agent systems presents a unique challenge not normally found in single-agent learning: the learning environment is non-stationary from the perspective of an individual agent. In this chapter, we were attempting to address this challenge and presented new multi-agent learning algorithms. As competitive learning scenarios can occur in cooperative MAS when we design local reward signals (instead of using the single global reward) for agents' learning, our MARL algorithms consider both

competitive and non-competitive scenarios. We introduced IGA-PP, a new gradient-based algorithm, by using policy prediction in basic gradient ascent. We proved that, in two-player, two-action, general-sum matrix games, IGA-PP in self-play or against IGA would lead players' strategies to converge to a Nash equilibrium. Based on the theoretical analysis of IGA-PP, we then proposed PGA-APP, a new practical MARL algorithm, only requiring the observation of the reward of selecting an action. Empirical results in normal-form games, distributed task allocation problem and network routing showed that PGA-APP converged faster and in a wider variety of situations than state-of-the-art MARL algorithms.

As shown in Figure 1.2, effective multi-agent learning algorithms are a key component for our MARL paradigm to developing effective local policies for agents in cooperative multi-agent systems. However, as shown in next chapter, MARL exploiting interaction locality alone may not always perform well in large systems. In the next chapter, we will present a supervision framework that coordinates multi-agent learning to improve the learning performance in large-scale cooperative multi-agent systems.

Part III

COORDINATING MULTI-AGENT LEARNING

CHAPTER 6

AN ORGANIZATIONAL CONTROL FRAMEWORK FOR COORDINATING MULTI-AGENT LEARNING

As shown in previous chapters, multi-agent reinforcement learning (MARL) techniques potentially provides an approximate, scalable approach to developing distributed coordination policies for agents in cooperate multi-agent systems. In order to achieve scalability of MARL [2, 24, 130], as we did for our application examples discussed in previous chapters, the learning of each agent has been restricted to using information received only from its immediate neighbors to update its estimates of the world states (i.e., Q-values for state-action pairs). However, this constraint results in long latency as state information propagates to agents further away. Such latency can result in neighborhood information being outdated, leading to mutually inconsistent views among agents. As a result, such a limited view for each agent and the non-stationarity of the environment (all agents are simultaneously learning their own policies) causes MARL algorithms to converge slowly and even diverge in some cases. The slowness of MARL convergence is further degraded by the large policy search space of each agent. Each agent's policy not only includes its local state and actions but also some characteristics of the states and actions of its neighboring agents [2], or the state size of each agent may be proportional to the number of agents in the system [24]. In this paper, we will present a supervision framework for coordinating MARL to address these challenges. The work of this chapter was published in AAMAS 2009 [123].

Our supervision framework, called Multi-Agent Supervisory Policy Adaptation (MASPA), employs low-overhead organizational control to guide multi-agent learning and accelerate

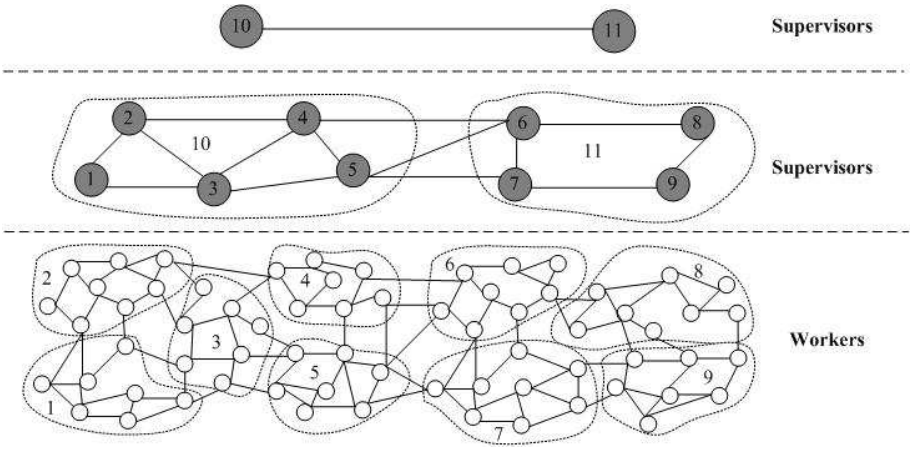


Figure 6.1. An organizational structure for multi-level supervision

its convergence. MASPA is composed of three components: a multi-level supervision organization (a meta-organization built on top of the agents' overlay network), a communication protocol for exchanging information between lower-level agents and higher-level supervising agents, and a policy adaptation mechanism that integrates organizational control information into MARL algorithms (e.g., those developed in previous chapters, GIGA [133], WPL [1], etc.) to guide the exploration process of each learning agent.

The key idea of MASPA is as follows. Each level in the supervision organization is an overlay network in itself. For example, Figure 6.1 shows a three-level supervision organizational structure. The abstracted states of lower-level agents travel upwards so that higher-level supervising agents can generate a broader view of the state of the network. This broader view comes from not only information about the states of lower-level agents but also information from neighboring supervising agents. In turn, this broader view results in creating supervisory information which is passed down the hierarchy. This supervisory information guides the learning of agents in collectively exploring their state-action spaces more efficiently, and consequently results in faster convergence. To provide up-to-date supervisory information, the process above is periodically repeated.

In this way, MASPA deals with scalability issues by using approximate partial global views that can be acquired with relatively low overhead. The use of these dynamic views

does not increase the state space of individual agent, but rather are used to generate directives for each agent so that its exploration is both more informed and more coordinated with other agents. To our knowledge, MASPA is the first framework that surrounds and coordinates multi-agent learning with organizational control. It has a hierarchy of control and data abstraction, which is conceptually different from existing hierarchical multi-agent learning algorithms that uses a hierarchy of task abstraction. In addition, MASPA can be used together with approaches that reduces the policy search space to further speed up the learning.

As other approaches to improving MARL algorithms, the use of MASPA requires some additional knowledge. This knowledge is used to decide what organizational structure needs to be formed, what abstracted state information is useful, and how to convert this information into supervisory information. However, MASPA itself is a general framework that dynamically guides the learning of agents. We verified the generality of MASPA with its applications in different domains (distributed task allocation and network routing) with different MARL algorithms. Experimental results show that it not only dramatically speeds up the rate of MARL convergence, but also increases its likelihood of convergence.

MASPA assumes agents will voluntarily share their state information. It also implicitly assumes the original multi-agent system can be formed into a nearly decomposable hierarchy [89] of at least one level. This assumption implies that if agents in the original MAS are far apart in spatial terms, their behaviors are also far apart in causal terms. For example, in Figure 6.1, knowing detailed information about agents in cluster 6 will not significantly affect the behaviors of agents in cluster 1. Our assumptions hold in many real cooperative systems. Sensor network is one example, where the whole system is designed to cooperate and usually decomposable according to proximity. Other examples include package routing in the Internet, peer-to-peer file sharing or information retrieval, and resource sharing in grid computing.

To focus on the essence of MASPA coordinating multiagent learning and isolate its impact on the system performance, this chapter uses pre-defined supervision organization structures. Supervision organizations can be dynamically formed during the learning through a bottom-up self-organization approach [127]. For simplicity, this chapter limits the discussion to the case where learning only happens at the bottom level and supervising agents use pre-specified heuristics to make decisions, but, in principle, MASPA does not restrict supervising agents learning their supervision policies.

The rest of the chapter is organized as follows: Section 6.1 presents a multi-level organizational structure for automated supervision mechanism. Section 6.2 defines a communication protocol for agents at different levels. Section 6.3 describe the supervisory policy adaptation that integrates supervisory information into MARL algorithms. Section 6.4 empirically evaluates our framework on distributed task allocation problem and network routing. Finally, Section 6.5 concludes this work and discusses some future work.

6.1 Organizational Supervision

Supervision mechanisms commonly exist in human organizations, such as enterprises and governments. The purpose of these mechanisms is to run an organization effectively and efficiently to fulfill the organization goals. Supervision involves gathering information, making decisions, and providing directions to regulate and coordinate actions of organization members. The practical effectiveness of supervision mechanisms in human organizations, especially in large organizations, inspired us to introduce a similar mechanism into multi-agent systems in order to improve the efficiency of MARL algorithms.

To add a supervision mechanism to a MAS with an overlay structure, MASPA adopts a multi-level, clustered organizational structure. Agents in the original overlay network, called workers, are clustered based on some measure (e.g., geographical distance). Each cluster is supervised by one agent, called the supervisor, and its member agents are called subordinates (note that subordinates at the lowest level are workers). The supervisor role

can be played by a dedicated agent or one of the workers. If the number of supervisors is large, a group of higher-level supervisors can be added, and so on, forming a multi-level supervision structure.[1] In this chapter, our discussion focuses on the situation where each agent belongs to only one cluster.

Two supervisors at the same level are adjacent if and only if at least one subordinate of one supervisor is adjacent to at least one subordinate of the other. Communication links, which can be physical or logical, exist between adjacent workers, between adjacent supervisors, and between subordinates and their supervisors. Figure 6.1 shows a three-level organizational structure. The bottom level is the overlay network of workers which forms 9 clusters. A shaded circle represents a supervisor, which is responsible for a corresponding cluster. Note that links between subordinates and their supervisors are omitted in this figure.

6.2 Communication Protocol

Each agent can demonstrate both fast and slow dynamics in how its features change. Fast dynamics of an agent are exhibited by the changes of such features as those that represent interactions with other agents, its local state, and its policy (or value function). Slow dynamics are exhibited by the changes of an agent's *abstracted state*. The abstract state is defined by a vector of features, which can be projected from features with fast dynamics by using such techniques as:

- Using partial components of a feature and ignoring other components that do not affect slow dynamics
- Using some statistics (e.g., mean, mode, etc.) of a feature generated over the temporal or spatial scale

[1] The top supervision level can have multiple supervisors.

- Replacing a fast-changing feature with its distribution parameters if its changes follow some statistical distribution

Similarly, each cluster also has fast and slow dynamics. Fast dynamics of a cluster are exhibited by that of its members. Slow dynamics of a cluster are captured by the changes of its supervisor's abstracted state. The abstracted state of a supervisor is projected either from the abstracted states of its subordinates or directly from features with fast dynamics of its subordinates. MASPA assumes that a supervisor can make rational decisions based on its own and neighbors' abstracted states.

MASPA uses three types of communication messages: *report*, *suggestion*, and *rule*. A report is used by a subordinate to pass its abstracted state upwards to provide its supervisor with a broader view. A supervisor also sends its report to its adjacent supervisors at the same level in addition to its immediate supervisor (if any). The supervisor's view is based on not only the agents that it supervises (directly or indirectly) but also its neighboring supervisors. This peer-supervisor communication allows each supervisor to make rational local decisions when directions from its immediate supervisor are unavailable.

Based upon this information, a supervisor employs its expertise, integrates directions from its superordinate supervisor, and provides supervisory information to its subordinates. Rules and suggestions are used to transmit supervisory information. We define a *rule* as a tuple $\langle c, F \rangle$, where

- c: a condition specifying a set of satisfied states
- F: a set of forbidden actions for states specified by c

A *suggestion* is defined as a tuple $\langle c, A, d \rangle$, where

- c: a condition specifying a set of satisfied states
- A: a set of actions
- d: the suggestion degree, whose range is $[-1, 1]$

A suggestion with a negative degree, called a *negative suggestion*, urges a subordinate not to do the specified actions. In contrast, a suggestion with a positive degree, called a *positive suggestion*, encourages a subordinate to do the specified action. The greater the absolute value of the suggestion degree, the stronger the impact of the suggestion on the supervised agent.

Each rule (or suggestion) contains a condition specifying states where it can be applied. Subordinates are required to obey rules from their supervisors. Rules are "hard" constraints on subordinates' behavior. In contrast, suggestions are "soft" constraints and allow a supervisor to express its preference for subordinates' behavior. A supervisor has a more global view but may lack detailed information about its subordinates' local policies and its own surrounding environment. Using suggestions, the supervisor is able to affect a subordinate's policy yet allow the subordinate to override its directives when needed. The implicit assumption is that a supervisor's suggestions will be correct most of the time so that the penalty of bad suggestions is outweighed by good suggestions. Therefore, a subordinate does not rigidly adopt suggestions. The effect of a suggestion on a subordinate's local decision making may vary, depending on its current policy and state. A supervisor will refine or cancel rules and suggestions as new or updated information becomes available.

A set of rules are in conflict if they forbid all possible actions on some state(s). Two suggestions are in conflict if one is positive and the other is negative and they share some state(s) and action(s). A rule conflicts with a suggestion if a state-action pair is forbidden by the rule but is encouraged by the suggestion. In our supervision mechanism, we assume each supervisor is rational and will not generate rules and suggestions that are in conflict. However, in a multi-level supervision structure, a supervisor's local decision may conflict with its superordinate (the supervisor's supervisor) direction. Rules have higher priority than suggestions. There are several strategies for resolving conflicts between rules or between suggestions, such as always taking its superordinate or local rule, stochastically selecting a rule, or requesting additional information to make a decision. The strategy

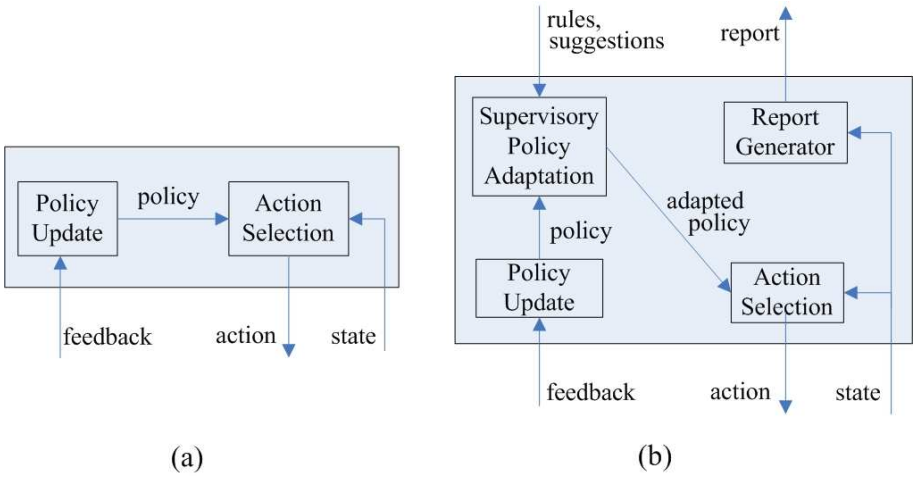


Figure 6.2. Unsupervised MARL vs. Supervised MARL with MASPA

choice depends on the application domain. Note that it may not always be wise to select the superordinate decision, because, although the superordinate supervisor has a broader view, its decision is based on abstracted information. The strategy used here for resolving conflicts picks the most constraining rule and combines suggestions by summing the degrees of the strongest positive suggestion and the strongest negative suggestion.

6.3 Supervisory Policy Adaptation

Using MARL, each agent gradually improves its action policy as it interacts with other agents and the environment. A *pure* policy deterministically chooses one action for each state. A *mixed or stochastic* policy specifies a probability distribution over the available actions for each state. A policy can be represented as a function $\pi(s, a)$, which specifies the probability that an agent will execute action a at state s. As argued in [94], mixed policies can work better than pure policies in partially observable environments, if both are limited to act based on the current percept. Due to partial observability, most MARL algorithms are designed to learn mixed policies. The rest of this section shows how mixed policy MARL algorithms can take advantage of higher-level information specified by rules and suggestions to speed up convergence.

As shown in Figure 6.2 (a), a typical MARL algorithm contains two components: policy (or action-value function) update and action selection based on the learned policy. One common method to speed up learning is to supply an agent with additional reward to encourage some particular actions, which is called reward shaping [72]. This use of the special reward affects both policy update and action selection. In a single-agent setting, there are potential function forms of reward shaping that leave the optimal policy/value-function unchanged [72]. However, due to the non-stationary learning environment in a multi-agent setting, reward shaping may generate a policy that is undesirable in that they may distract from the main goal, which is supported by the normal reward.

MASPA directly biases the action selection for exploration without changing the policy update process. As shown in Figure 6.2 (b), MASPA' supervisory policy adaptation integrates rules and suggestions into the policy learned by an unsupervised MARL algorithm and then outputs an adapted policy. This adapted policy is intended to control exploration. Our integration assumes policies learned by an unsupervised MARL are stochastic. The report generator computes the abstract state of the agent.

Let R and G be the rule set and suggestion set, respectively, that a worker received and π be its learned policy. We define $R(s,a) = \{\langle c, F\rangle \in R|$ state s satisfies the condition c and $a \in F\}$ and $G(s,a) = \{\langle c, A, d\rangle \in G|$ state s satisfies the condition c and $a \in A\}$. As we assume a supervisor is rational, it will not generate more than one suggestion for a subordinate that satisfies a state-action pair. Thus, $|G(s,a)| \leq 1$. The function $deg(s,a)$ that returns the degree of the satisfied suggestion is defined as following:

$$deg(s,a) = \begin{cases} 0 & \text{if } |G(s,a)| = 0 \\ d & \text{if } |G(s,a)| = 1 \text{ and } \langle c, A, d\rangle \in G(s,a) \end{cases}$$

Then the adapted policy π^A for the action selection is generated by the supervisory policy adaptation:

$$\pi^A(s,a) = \begin{cases} 0 & \text{if } R(s,a) \neq \emptyset \\ \pi(s,a) + \pi(s,a) * \eta(s) * deg(s,a) & \text{else if } deg(s,a) \leq 0 \\ \pi(s,a) + (1 - \pi(s,a)) * \eta(s) * deg(s,a) & \text{else if } deg(s,a) > 0 \end{cases}$$

The state-dependent function $\eta(s)$ ranges from $[0, 1]$. As similarly defined in the supervised actor-critic architecture [81], it determines the receptivity for suggestions and allows the agent to selectively accept suggestions based on its current state. For instance, if an agent becomes more confident in the effectiveness of its local policy on state s because it has more experience with it, then $\eta(s)$ decreases as learning progresses. In our experiments, we set $\eta(s) = k/(k + visits(s))$ where k is a constant and $visits(s)$ returns the number of visits on the state s.

With the supervisory policy adaptation, a rule explicitly specifies undesirable actions for some states and is used to prune the state-action space. Suggestions, on the other hand, are used to bias agent exploration. To integrate suggestions into MARL, MASPA uses the strategy that the lower the probability of a state-action pair, the greater the effect a positive suggestion has on the pair and the less the effect a negative suggestion has on it. The underlying idea is intuitive. If the agent's local policy already agrees with the supervisor's suggestions, as indicated by the policy having high (or low) probabilities for state-action pairs from the positive (or negative) suggestions, it is going to change its local policy very little (if at all); otherwise, the agent follows the supervisor's suggestions and makes a more significant change to its local policy.

To normalize π^A such that it sums to 1 for each state, the *limit* function from GIGA [133] is applied with minor modifications so that every action is explored with minimum probability ϵ:

$$\pi^A = limit(\pi^A) = argmin_{x:valid(x)}|\pi^A - x|$$

i.e., $limit(\pi^A)$ returns a valid policy that is closest to π^A.

Our normalization also implicitly solves the issue of rules in conflict. If a set of rules forbids all actions on a state, then the probability of each action is set to 0. After normalization, the probabilities of all actions are equal, that is, the action choice becomes completely random. This strategy is reasonable when the agent does not know the consequence of violating each rule.

6.4 Experimental Results

We have tested MASPA in two different domains: distributed task allocation problem (DTAP) and network routing. In the following experiments, we manually cluster agents in the overlay network using Manhattan distance. The agent closest to the center of each cluster is elected as the supervisor. Supervisors also play the worker role. We assume there are links that allows direct communication between subordinates and their supervisors and between adjacent supervisors.

6.4.1 Distributed Task Allocation

We evaluated MASPA in a simplified DTAP [2] with Poisson task arrival and exponential service time. Agents are organized in an overlay network. Each agent receives tasks from the environment at a certain rate. At each time unit, an agent makes a decision for each task received during this time unit whether to execute the task locally or send it to a neighboring agent for processing. A task to be executed locally will be added to the local queue with unlimited queue length, where tasks are executed on a first-come-first-serve basis. Agents interact via communication messages and communication delay between two agents is proportional to the distance between them, one time unit per distance unit. The main goal of DTAP is to minimize the total service time of all tasks, averaged by the number of tasks, $ATST = \frac{\sum_{T \in \bar{T}_\tau} TST(T)}{|\bar{T}_\tau|}$, where $\bar{T}_\tau$ is the set of tasks received during a time period τ and $TST(T)$ is the total service time that task T spends in the system, which includes the routing time, queuing time, and execution time.

6.4.1.1 Experimental Setup

We chose one representative MARL algorithm, the Weighted Policy Learner (WPL) algorithm [1], for each worker to learn task allocation policies, and compared its performance with and without MASPA. WPL is a gradient ascent algorithm where the gradient is weighted by $\pi(a)$ if it is negative; otherwise, it will weighted by $(1 - \pi(a))$. So effectively, the probability of choosing a good action increases by a rate that decreases when the probability approaches to 1. Similarly, the probability of choosing a bad action decreases by a rate that decreases when the probability approaches to 0. A worker's state is defined by the current work load (or total work units) in the local queue.

The abstracted state of a worker is projected from its states and defined by its average work load over a period of time τ ($\tau = 500$ in our experiments). The abstracted state of a supervisor is defined by the average load of its cluster, which can be computed from the abstracted states of its subordinates. A subordinate sends a report, which contains its abstracted state, to its supervisor every τ time period. Supervisors use simple heuristics to generate rules and suggestions. With an abstracted state $\langle \bar{l} \rangle$, a supervisor generates a rule that specifies, for all states whose work load exceeds $\bar{l}$, a worker should not add a new task to the local queue. This rule helps balance load within the cluster. A supervisor also generates positive (or negative) suggestions for its subordinates to encourage (or discourage) them forwarding more tasks to a neighboring cluster that has a lower (or higher) average load. The suggestion degree for each subordinate depends on the difference between the average load of two clusters, the number of agents on the boundary, and the distance of the subordinate to the boundary. Therefore, suggestions are used to help balance the load across clusters.

Three measurements are evaluated: the average total service time (ATST), the average number of messages (AMSG) per task, and the time of convergence (TOC). ATST indicates the overall system performance, which can reflect the effectiveness of learning and supervision mechanism and can also be used to verify system stability (convergence) by

showing a monotonic decrease in ATST as agents gain more experiences. AMSG shows the overall communication overhead for finishing one task, which including both for task routing and MASPA supervision. To calculate TOC, we take sequential ATST values with certain size and then calculate the ratio of those values' deviation to their mean. If the ratio is less than a threshold (e.g., 0.025), then we consider the system stable. TOC is the start time of the selected points.

Experiments were conducted using uniform two-dimension grid networks of agents with different sizes: 6x6, 10x10, and 27x27, all of which show similar results. But as the size of the system increases, the MASPA impact on the system performance becomes greater. For brevity, we only present here the results for the 27x27 grid (with 729 agents). For simplicity, we assume that all agents have the same execution rate and that tasks are not decomposable. The mean of task service time is $\mu = 10$. We tested three patterns of task arrival rates:

Uneven Center Load where 121 agents in the centric 11x11 grid receive tasks and other agents receive no tasks from the external environment. In the centric 11x11 grid, the task arrival rate of agents on the outermost 6 columns is $\lambda = 0.8$ and the rate of the rest agents is $\lambda = 0.2$.

Corner Load where only agents in the 12x12 grid at the up-left corner receive tasks from the external environment. In that 12x12 grid, the agents in the 9x9 grid at the up-left corner has the task arrive rate $\lambda = 0.2$ and the rest agents has the rate $\lambda = 0.7$.

Boundary Load where the 200 outermost agents receive tasks with rate $\lambda = 0.33$ and other agents receive no tasks from the external environment.

In each simulation run, ATST and AMSG are computed every 500 time units to measure the progress of the system performance. Results are then averaged over 10 simulation runs and the variance is computed across the runs. All agents use WPL with learning rate 0.001. Our experiments use the parameter $\eta(s) = 1000/(1000 + visits(s))$.

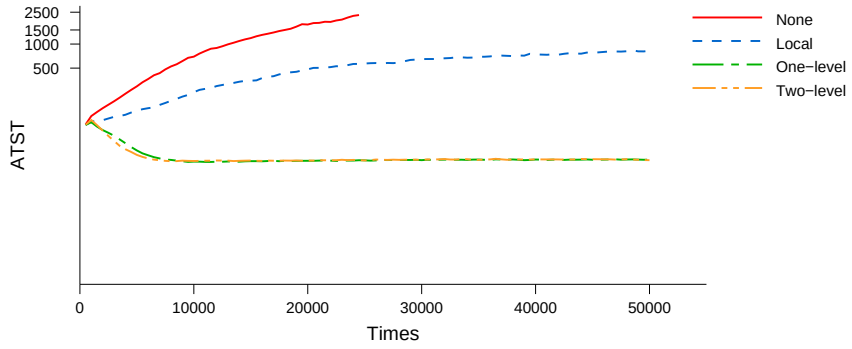


Figure 6.3. ATST for different structures with uneven center load

We compared four structures: *no supervision, local supervision, one-level supervision, and two-level supervision*. In the *local supervision* structure, agents are their own supervisors. With this structure, each agent gains a view only about itself and its neighbors, which is not much different from its view in the organization without supervision. We use the *local supervision* structure to evaluate whether domain knowledge combined with a limited view, which is used to create rules and suggestions, still improves the system performance. In contrast, the performance of the two following structures with supervision show the benefits of having a broader view combined with domain knowledge. The *one-level supervision* structure has 81 clusters, each of which is a 3x3 grid and the agent at each cluster center is elected as the supervisor. The *two-level supervision* structure forms from the *one-level supervision* structure by grouping 81 supervisors into 9 clusters, each of which is a 3x3 grid. The supervision structures with three or more levels did not show further improvement over the two-level supervision in our DTAP experiments. This is because a wide-range task transfer causes a long routing time which offsets the reduction of the queuing time in each agent.

6.4.1.2 Results & Discussions

Figure 6.3, 6.4 and 6.5 plot the trend of ATST, as agents learn, for different organization structures with different task arrival patterns. Note that the y axis in the plots is logarithmic.

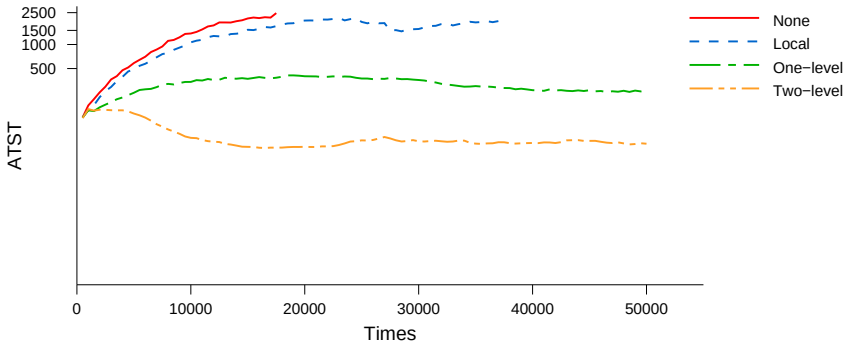


Figure 6.4. ATST for different structures with corner load

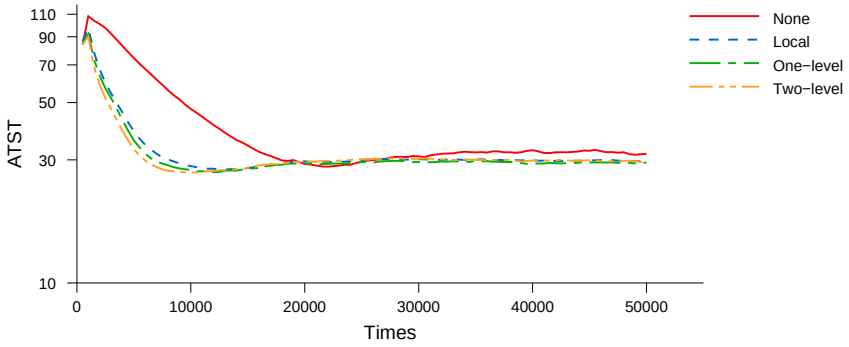


Figure 6.5. ATST for different structures with boundary load

As expected, MASPA improves both the likelihood and speed of the learning convergence. The broader the view MASPA observes, the greater the system performance it improves. In addition, several other observations are also noted.

Under both uneven center load and corner load, the system without MASPA does not seem to converge. From Figure 6.3 and 6.4, we see that both simulations ends before 50000 time units. This happens because, using random exploration, agents in the grid do not learn and propagate quickly enough knowledge about where light-loaded agents are. As a result, for example, under uneven center load patter, more and more tasks loop and reside in the center 11x11 grid where agents receive external tasks. This makes the system load severely unbalanced and the system capability not well utilized, which causes the system load to monotonically increase. Our simulations ran out of all computing resources and terminated before showing any signs of convergence. In contrast, observing broader views, MASPA guides and coordinates the exploration of agents and allows them to learn quickly to effectively route tasks.

Under both uneven center load and corner load, *local supervision* does not prevent system divergence. This is because uneven task arrival rates in both patterns cause many agent's local view of the system to become inconsistent with the global system view. For example, under uneven center load pattern, many overloaded agents at the center columns find their neighbors having even higher loads. As a result, *local supervision* generates incorrect directives for them to explore their actions. For similar reasons, explained at a cluster level instead of a worker level, the system with *one-level supervision* doesn't perform well under corner load pattern.

Broader views for MASPA do not necessarily significantly improve the system performance. For example, under uneven center load, *one-level supervision* and *two-level supervision* show similar performance, and, under boundary load pattern, all supervision structures demonstrate similar performance. This is because, in both cases, broader views do not provide much additional information for MASPA. For example, under the bound-

ary load pattern, local work loads in the whole network quickly form some pattern, where an agent farther away from the network boundary usually has a lighter local load. Then, based on their local view, most agents generate suggestions for themselves to forward tasks to neighbors closer to the network center, which are coincidentally similar to suggestions generated from a broader view (e.g., one-level or two-level supervision).

Supervision	ATST	AMSG	TOC
None	N/A	N/A	N/A
Local	N/A	N/A	N/A
One-level	33.41 ± 0.66	10.21 ± 0.25	7500
Two-level	34.08 ± 0.62	10.60 ± 0.22	6000

Table 6.1. Performance of different structures with uneven center load

Supervision	ATST	AMSG	TOC
None	N/A	N/A	N/A
Local	N/A	N/A	N/A
One-level	265.50 ± 6.59	24.83 ± 1.34	38500
Two-level	51.37 ± 0.88	16.33 ± 0.26	14000

Table 6.2. Performance of different structures with corner load

Supervision	ATST	AMSG	TOC
None	29.26 ± 0.71	6.90 ± 0.21	17500
Local	28.21 ± 0.59	7.02 ± 0.09	8500
One-level	27.64 ± 0.60	6.94 ± 0.16	7500
Two-level	27.49 ± 0.60	7.14 ± 0.14	6500

Table 6.3. Performance of different structures with boundary load

Table 6.1, Table 6.2, and Table 6.3 show the different measures for each supervision structure at their own convergence time point. In addition to increasing the convergence rate, MASPA also decreases the system ATST. In most cases, the broader the views MASPA observes, the lower the ATST the system generates. We can also observe that MASPA does not incur heavy communication overhead. For example, with the boundary load pattern,

one-level supervision has less than 0.6% communication overhead. With the corner load pattern, *two-level supervision* actually produces lower AMSG than *one-level supervision*. This is because *two-level supervision* leads workers to learn more quickly and effectively to forward tasks to the right workers, which dramatically reduces the number of messages for routing tasks and offsets the overhead from an additional level of supervision.

During the experiments, we observed that supervisory information corresponding to coarse-grained control tend to be more helpful than that corresponding to fine-grained control in improving the system performance. Moreover, fine-grained may even decrease system performance. Coarse-grained control considers and operates on the whole cluster as one entity, while fine-grained control operates on individual cluster members. "Moving more tasks from my cluster to one of neighboring clusters" and "balancing the load within the cluster" are examples of coarse-grained control . "Moving more tasks from a high-loaded agent to a low-loaded agent along the shortest path" is an example of fine-grained control. One explanation for this observation is that supervisory information corresponding to coarse-grained control results in more coordination among agents' exploration, speeding up the learning convergence. In contrast, in our simulation, due to lack of detailed information of each cluster member, fine-grained control for some individual members is not able to fully evaluate the impact on and from other agents. As a result, the fine-grained control may interfere with the normal learning process of other agents and the dynamics of other agents may degrade the fine-grained control.

We have explored different values of cluster size and found that system performance decreases with cluster size that are either too small (e.g., ≤ 5) or too large (e.g., ≥ 100). This is because, with too small a cluster size, supervisors do not collect enough information to create correct rules and suggestions. When a cluster size is too large, the representation of cluster abstracted states for DTAP (i.e. averaging loads of subordinates) ignores the variance among subordinates. As a result, supervisors are not able to create proper rules and suggestions for every subordinate. Therefore, there is a trade-off for the cluster size.

In addition, cluster sizes that produce the best performance vary in different environments (e.g., different task arrival patterns).

Similarly, there is a trade-off in the length of the report period. A too short report period causes a large variance of the abstracted state (also increases communication overhead) and results in oscillating suggestions and rules. A too long report period causes the supervisory information received by workers to be out-dated and as a result, decreases the convergence rate.

6.4.2 Network Routing

We also evaluated our framework using a network routing simulator adopted from Boyan and Littman [24]. It is a discrete time simulator of communication networks with various topologies. A communication network consists of a homogeneous set of nodes (or agents) and links between them. Packets are periodically introduced into the network under a Poisson distribution with a random origin and destination. No two packets have the same agent as their origin and destination. When a packet arrives at an agent, the agent puts it into the local FIFO (first in first out) queue. At each time step, an agent makes its routing decision to forward the top packet in the queue to one of its neighbors. Once a packet reaches its destination, it is removed from the network. In our experiments, we set the time cost of sending a packet down a link as a unit cost. So the delivery time of packet consists of its transmission cost and its waiting time in queues. The main goal of a network routing algorithm for this problem is to minimize the Average Delivery Time (ADT) of all packets.

6.4.2.1 Experimental Setup

Each agent uses a Policy Gradient Descent (PGD) algorithm to learn its routing policies. The PGD algorithm is a variant of the GIGA algorithm [133], which minimizes the total discounted cost and approximates the policy gradient of each state-action pair with the normalized difference of its Q-value and the expected Q-value on that state. PGD learns stochastic policies, but, unlike multi-agent OLPOMDP [103] and GAPS [77] that were also

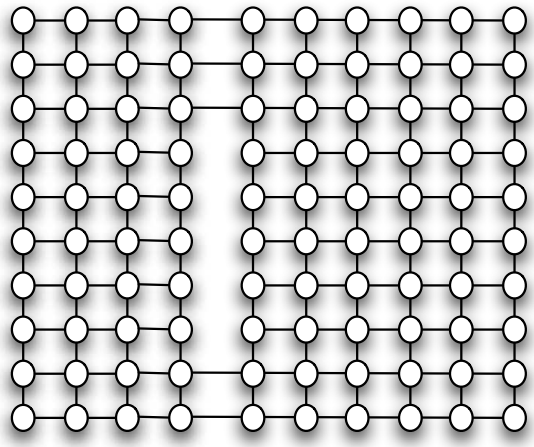


Figure 6.6. The 10 x 10 grid topology

applied to network routing problem, it does not require a global reward signal. The state s is defined by the destination of the packet that an agent is forwarding. We define $Q_x(s, a)$ as the estimated time that an agent x takes to deliver a packet to the destination s through its neighbor a, including any time that the packet would have to spend in the agent x's queue. The "cost signal" $r(s, a)$ for forwarding a packet with destination s to its neighbor a is $q_a + w + t$, where w is the waiting time of the packet in x's queue and t is the transmission time between agent x and a. The Q-learning algorithm is used to update x's estimates.

The MASPA implementation in network routing is similar to that in DTAP. The main difference is the way that MASPA messages are generated. In the network routing problem, we do not use rules. The abstracted state of a worker (or supervisor) is defined a vector $\langle t_1, t_2, \ldots, t_m \rangle$, where t_i is the average estimated time that the worker (or the supervisor's cluster) takes to deliver a packet to destination agents in cluster i. So, by using statistic *mean*, the abstract state of a worker can be computed from its Q-value table and a supervisor's abstracted state can be projected from its subordinates' abstracted states. A simple heuristic is used for generating suggestions. A supervisor always produces positive (or negative) suggestions for its subordinates to encourage (or discourage) them forward-

ing packets to clusters with lower (or higher) estimated delivery time to some destination cluster. The suggestion degree for each subordinate depends on the difference between the average estimated delivery time of neighboring clusters and the distance of the subordinate to the boundary.

We have tested the PGD algorithm with and without MASPA on several network topologies with various number of nodes, all of which show similar results. For brevity, we concentrate on the result analysis for the 10 x 10 grid network pictured in Figure 6.6. The Q-routing [24] algorithm is used as baseline, which learns deterministic policies. Two measurements are evaluated: the average delivery time (ADT) and the time of convergence (TOC). The ADT is computed every 1000 time units. To calculate TOC, we take 50 sequential ADT values and then calculate the ratio of those values' deviation to their mean. If their mean is less than the maximum expected ADT (we use 300) and the ratio is less than a threshold (we use 0.05), then we consider the system stable. TOC is the start time of the selected points.

Results are then averaged over 10 simulation runs. All agents use the PGD algorithm with a learning rate $\zeta = 0.1$. Workers send reports to their supervisors every 500 time units. Our experiments use the parameter $\eta(s) = 20000/(20000 + visits(s))$.

6.4.2.2 Results & Discussions

Figure 6.7 shows the performance trend as agents learn under network load$= 7.0$. The network load is the average number of packages entering the network at each time unit. All three algorithms, after initial periods of inefficiency during which they randomly explore the environment, gradually improve their performance and stabilize. At the very early period, MASPA does not improve the performance much. This is because, due to almost complete random exploration, subordinates do not provide accurate environment information to their supervisor, which may result in some improper suggestions. As information

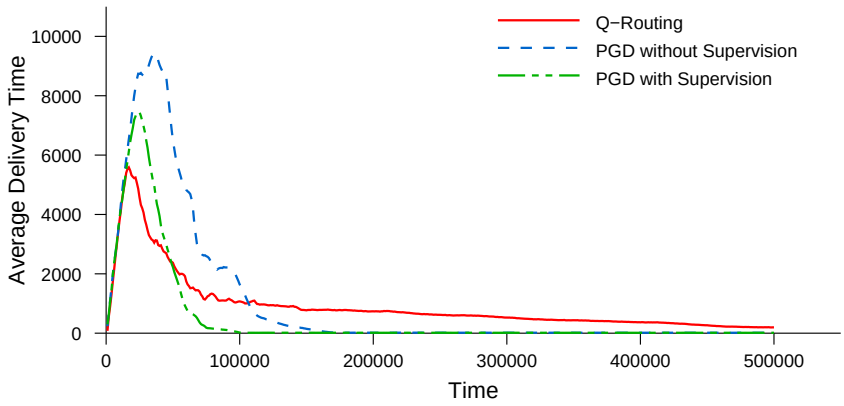


Figure 6.7. Performance under network load = 7.0

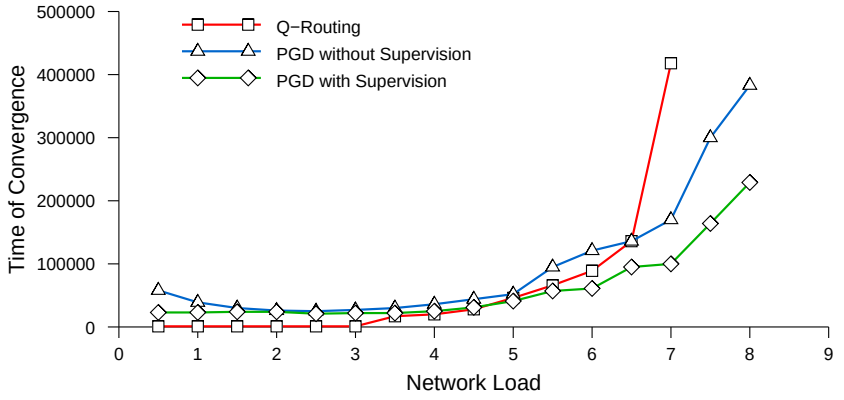


Figure 6.8. Time of Convergence at various loads

accuracy increases, MASPA properly biases the policy search of the PGD algorithm and speeds up the convergence. Due to policy oscillation, Q-routing shows slow convergence.

Figure 6.8 shows the TOC of three algorithms under various network loads. As expected, MASPA consistently speeds up the convergence of the PGD algorithm. The higher the network load, the greater the speed improvement. For example, when load ≥ 5.5, MASPA decreases the TOC by around 40% or more. Under low network loads, optimal policies usually follows shortest paths, so they are deterministic. The PGD algorithms use gradient update and gradually converge to deterministic policies, slower than Q-routing that

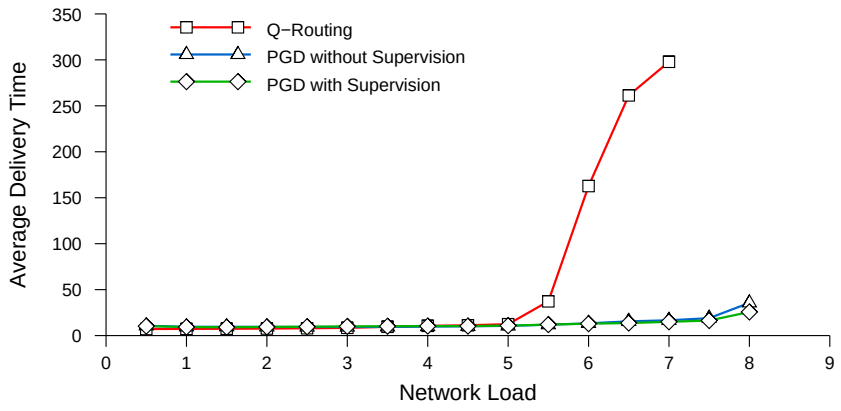


Figure 6.9. Delivery time at various loads

directly learns deterministic policies. However, under high loads, where optimal policies are usually stochastic, the Q-routing policies show oscillation during the learning and the PGD algorithm with MASPA converges faster to stochastic policies.

Figure 6.9 shows the ADT at the convergence time point under various network loads. Under low loads, as both PGD algorithms, with and without MASPA, converge to deterministic policies, they show almost the same performance. Due to random exploration with some probability, they perform slightly worse than Q-routing. However, under high loads, MASPA improves the PGD performance. For example, when load ≥ 6.5, MASPA decreases the ADT by at least 10%, and when load $= 8.0$, MASPA reduces the ADT by around 30%. As both PGD algorithms converge to stochastic policies, which allows agents to simultaneously exploit multiple paths to deliver packets to a single destination, they perform much better than Q-routing under high loads.

6.5 Summary

In this chapter, we presented MASPA, a distributed supervision framework, that enables efficient learning in large-scale multi-agent systems. In MASPA, the automated supervision mechanism fuses activity information of lower-level agents and generates supervisory

information that guides and coordinates agents' learning process. This supervision mechanism continuously interacts with and dynamically controls the learning process. Simulation results obtained in two different domains with different MARL algorithms verified the generality of MASPA and demonstrated that MASPA significantly accelerates the learning process with relatively low communication overhead.

To facilitate this supervision framework to be applied to practical MARL applications, in next two chapters, we will attempt to address two important aspects of this framework: finding supervisory organizations for coordinating agents' learning processes and automating supervision process to dynamically generate supervisory information with little or no domain knowledge.

CHAPTER 7

SELF-ORGANIZATION FOR COORDINATING MULTI-AGENT LEARNING

In our previous chapter, we present a general supervision framework for addressing challenges of scaling MARL to large complex MAS applications. This framework employs low-overhead, multi-level organizational control to dynamically coordinate agents' learning processes in order to improve the speed, likelihood, and quality of their learning convergence. One important question arising from this supervision framework is how to find a proper supervisory organization for coordinating MARL, which will be addressed in this chapter. More specifically, we will answer the following questions: can supervisory organizations automatically form while agents are concurrently learning their decision policies? do such dynamically evolving organizations perform better than static supervisory organizations? This chapter will present a distributed self-organization approach [128] to automatically and incrementally form supervisory organizations for better coordinating agent's learning processes while they are concurrently learning their decision policies. The work of this chapter was published in AAMAS 2010 [128].

This chapter makes a twofold contribution. First, we formalize joint-event-driven interactions among agents using a DEC-MDP model and define a measure for capturing the strength of such interactions. Second, we develop a distributed self-organization approach, based on the interaction measure, that dynamically adapts supervision organizations for coordinating decentralized reinforcement learning (DRL) during the learning process. Unlike the work in [2], our self-organization process does not change the connectivity of the original agent network, but form a hierarchical supervisory organization on top of it. The key problem of the organization adaptation is to decide which agents need to be clustered

together so that their exploration strategies can be coordinated. Our approach to this problem is inspired by the concept of *nearly decomposable systems* [89], where interactions between subsystems are generally weaker than interactions within subsystems. In order to improve the quality and reduce the complexity of coordinating DRL, our approach attempts to group agents together that strongly interact with each other. Unlike most of the previous work on self-organization (e.g., [35, 91]), our approach uses dynamic, rather than static, information about agents' behaviors based on their current state of learning. In our approach, the organization adaptation and individual agents' learning concurrently progress and interact with each other. Experimental results show that our dynamically evolving organizations outperform predefined organizations for coordinating DRL.

The rest of the chapter is organized as follows. Section 7.1 reviews some background knowledge. Section 7.2 develops a distributed self-organization approach for dynamically evolving supervisory organizations to better coordinate DRL, and extends the supervision framework proposed in previous chapter to integrate our approach. Section 7.3 empirically evaluates our approach. Finally, Section 7.4 summarizes the contribution of this work.

7.1 Background

This section reviews a DEC-MDP model for representing collaborative MAS, DRL for learning efficient approximate policies for agents in collaborative MAS, and the supervision framework for improving the performance of DRL.

7.1.1 Average-Reward, Factored DEC-MDP

As discussed in Section 2.3, we will learn memoryless stochastic policies that mapping the immediate observation to an action in DEC-POMDP, which, in some sense, we assume that the global state of a DEC-POMDP is factored and each agent can fully observe its local state. In addition, since it is usually infeasible or not scalable to use the global reward signal for learning in a DEC-POMDP, we assume local reward signals exist that are specifi-

cally tailored and efficiently computable for individual agents. Therefore, we the following use factored DEC-MDP to model the multiagent sequential decision-making problem in a collaborative MAS.

Definition 11. *An n-agent factored DEC-MDP is defined by a tuple $\langle S, A, T, \mathcal{R} \rangle$, where*

- $S = S_1 \times \cdots \times S_n$ *is a finite set of world states, where S_i is the state space of agent i*
- $A = A_1 \times \cdots \times A_n$ *is a finite set of joint actions, where A_i is the action set for agent i*
- $T : S \times A \times S \to \Re$ *is the transition function. $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the probability of transiting to the next state $\mathbf{s}'$ after a joint action $\mathbf{a} \in A$ is taken by agents in state $\mathbf{s}$*
- $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ *is a set of reward functions. $R_i : S \times A \to \Re$ provides agent i with an individual reward $\mathbf{r_i} \in R_i(\mathbf{s}, \mathbf{a})$ for taking action $\mathbf{a}$ in state $\mathbf{s}$. The global reward is a weighted sum of all local rewards: $R(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} w_i R_i(\mathbf{s}, \mathbf{a})$, where w_i is a positive weight.*

A policy $\pi : S \times A \to \Re$ is a function which returns the probability of taking action $\mathbf{a} \in A$ for any given state $\mathbf{s} \in S$. Similar to [78], the value function for a policy π is defined relative to the average expected reward per time step under the policy:

$$\rho(\pi) = \lim_{N \to \infty} \frac{1}{N} \mathbf{E}[\sum_{t=0}^{N-1} R(\mathbf{s}^t, \mathbf{a}^t) | \pi] \tag{7.1}$$

where the expectation operator $\mathbf{E}(\cdot)$ averages over stochastic transitions and $\mathbf{s^t}$ and $\mathbf{a}^t$ are the global state and the action taken at time t, respectively. The optimal policy is a policy that yields the maximum value $\rho(\pi)$.

Assume that the Markov chain of states under policy π is ergodic. The expected reward $\rho(\pi)$ then does not depend on the starting state. Let $p(\mathbf{s}|\pi)$ be the probability of being in

state $\mathbf{s}$ under the policy π, which can be calculated as the average probability of being in state $\mathbf{s}$ at each time step over the infinite execution sequence:

$$p(\mathbf{s}|\pi) = \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} P(\mathbf{s}^t = \mathbf{s}) \tag{7.2}$$

Lemma 6. *Suppose $R(\mathbf{s})$ is the global reward function. Then the value of policy π is*

$$\rho(\pi) = \sum_{\mathbf{s}\in S} p(\mathbf{s}|\pi) \sum_{\mathbf{a}\in A} \pi(\mathbf{s},\mathbf{a}) R(\mathbf{s},\mathbf{a}) \tag{7.3}$$

The lemma follows immediately from Equation 7.2 and the definition of the policy value in Equation 7.1 based on the assumption that the state process is ergodic.

7.1.2 Decentralized Reinforcement Learning

DRL is used by agents to learn efficient approximate policies in a factored DEC-MDP environment, especially when the transition and reward function is unknown. Each agent learns its local policy based on its local observation and reward in presence of other agents, who are also learning a policy under the same conditions. The local policy $\pi_i : S_i \times A_i \to \Re$ for agent i returns the probability of taking action $a_i \in A_i$ in local state $s_i \in S_i$. As each agent only observes local reward signals, the value function of a local policy π_i of agent i is defined as:

$$\rho_i(\pi_i) = \lim_{N\to\infty} \frac{1}{N} \mathbf{E}[\sum_{t=0}^{N-1} r_i^t | \pi_i] \tag{7.4}$$

where the expectation operator $\mathbf{E}(\cdot)$ averages over both stochastic transitions and nondeterministic rewards and r_i^t is the local reward received at time t. The local reward $r_i^t = R_i(\mathbf{s^t})$ depends on the global state $\mathbf{s^t}$ and appears nondeterministic from the local perspective. The objective of agent i is to learn an optimal policy π_i^* to maximize $\rho_i(\pi_i)$.

Similar to Lemma 6, we can also reformulate the value function of the local policy.

Lemma 7. *Suppose* $\mathbf{E}[r_i(s_i)|\pi]$ *is the expected local reward of taking action* a_i *in state* s_i *given a joint policy* π.

$$\rho_i(\pi_i|\pi_{-i}) = \sum_{s_i \in S_i} p(s_i|\pi) \sum_{a_i \in A_i} \pi_i(s_i, a_i)\mathbf{E}[r_i(s_i, a_i)|\pi], \tag{7.5}$$

where $p(s_i|\pi)$ *as the probability of being in local state* s_i *under the joint policy* π *and* π_{-i} *is the set of policies of all agents except agent* i.

Due to factored reward, we have the following lemma that can directly be proved from the definitions of factored DEC-MDP and value functions of both joint and local policies.

Lemma 8. *The value of a joint policy is a weighted sum of the values of local policies, that is,*

$$\rho(\pi) = \sum_i w_i \rho_i(\pi_i|\pi_{-i}), \tag{7.6}$$

where the joint policy $\pi = (\pi_1, \ldots, \pi_n)$ *and* π_{-i} *is the set of policies of all agents except agent* i.

7.2 Supervisory Organization Formation

In our previous chapter, we present a supervision framework that is intended to improve the speed, quality, and likelihood of DRL convergence. This framework employed low-overhead, periodic organizational control to coordinate and guide agents' exploration during the learning process. This section describes our approach to dynamically evolving a hierarchical supervisory organization for better coordinating DRL when agents are concurrently learning their decision policies. Organization formation is best described via answering two questions: how agent clusters are formed, and how a cluster supervisor is selected. Our approach adopts a relatively simple strategy for supervisor selection. Each cluster selects an agent as its supervisor that minimizes the communication overhead between supervisors and their subordinates. A new supervisor then establishes connections to supervisors of neighboring clusters based on the connectivity of their subordinates.

Agent clustering is to decide what agents should be grouped together so that their learning exploration strategies can be better coordinated by one supervisor. Because of limited resources of computation and communication, it is usually not feasible to put all agents together and use a fully centralized coordination mechanism. To deal with bounded resources and maintain satisficing performance of coordination, our clustering strategy is to cluster highly interdependent agents together, whose interactions have a great impact on the system performance, and meanwhile to minimize interactions across clusters. Thus the resulting system has a nearly decomposable, hierarchical structure, which reduces the complexity of coordinating DRL in a distributed way.

To measure the interdependency between agents, we characterize a type of interactions among agents, called *joint-event-driven interactions*, in a DEC-MDP model. We also define a measure for the strength of such interactions, called *gain of interactions*, and analyze how interactions between agents contribute to the system performance by using this measure. Based on this measure, we then propose a distributed, negotiation-based agent clustering algorithm to form a nearly decomposable organization structure. Finally, we discuss how to extend supervision framework proposed in [123] to integrate our self-organization approach. For clarity, this chapter focuses the discussion on forming a two-level hierarchy. Our organization formation approach can be iteratively applied in order to form a multi-level hierarchy.

7.2.1 Joint-Event-Driven Interactions

Definition 12. *A* **primitive event** $e_j = \langle s_j, a_j \rangle$ *occurs when agent* j *executes action* a_j *in state* s_j*. A* **joint event** $\vec{e}_X = \langle e_{j_1}, e_{j_2}, \ldots, e_{j_h} \rangle$ *contains a set of primitive events generated by agents* $X = \{j_1, j_2, \ldots, j_h\}$*. A joint event* $\vec{e}_X$ *occurs iff all of its primitive events occur.*

Note that our definition of a joint event is different from that of an event in [13], where an event occurs if any one of its primitive events occurs. For brevity, events discussed in this chapter refer to joint events. An event is used to capture the fact that some agents

did some specific activities. A primitive event can be generated by either an agent or the external environment. For convenience, we treat the external environment as an agent.

Definition 13. *A* **joint-event-driven interaction** $i_j^X = \langle \vec{e}_X, e_j \rangle$ *from a set of agents* X *onto agent* j *is a tuple that includes a joint event* $\vec{e}_X$ *and a primitive event* e_j*. A joint-event-driven interaction* i_j^X *is* **effective** *iff the event* $\vec{e}_X$ *affects the distribution over the resulting state of event* e_j*, that is,* $\exists s_j \in S_j$ *such that* $p(s_j^{t+1} = s_j | e_j^t = e_j) \neq p(s_j^{t+1} = s_j | e_j^t = e_j, \vec{e}_X^t = \vec{e}_X)$*, where* t *is the time.*

Here we define an interaction between agents as an affecting relationship, which is uni-directional. An effective interaction on an agent basically changes its transition function. If there exists an effective interaction $\langle \langle e_X \rangle, e_j \rangle$, then we say that agents X effectively interact with agent j.

Now we define a measure for the strength of interactions among agents. Let $E_X^j = \{\vec{e}_X | \exists e_j \in S_j \times A_j$ such that interaction $\langle \vec{e}_X, e_j \rangle$ is effective$\}$ be all joint events generated by a set of agents X that effectively interact with agent j.Let $V_j(s_j|\pi) = \sum_{a_j} \pi_j(s_j, a_j) \mathbf{E}[r_j(s_j, a_j)|\pi]$ be the expected value of being in state s_j, where π_j is the policy of agent j, and $\mathbf{E}[r_j(s_j, a_j)|\pi]$ is the expected reward of executing action a_j in state s_j.

Definition 14. *The* **gain of interactions** *from a set of agents* X *to agent* j*, given a joint policy* π*, is*

$$g(X, j|\pi) = \sum_{\vec{e}_X \in E_X^j} p(\vec{e}_X|\pi) \sum_{s_j} p(s_j|\vec{e}_X, \pi) V_j(s_j|\pi),$$

where $p(\vec{e}_X|\pi)$ *is the probability that event* $\vec{e}_X$ *occurs and* $p(s_j|\vec{e}_X)$ *is the probability of being in state* s_j *after* $\vec{e}_X$ *occurs.*

The value of the gain of interactions is affected by two factors: how frequently agents effectively interact (reflecting on $p(\vec{e}_X|\pi)$) and how well they are coordinated (reflecting on $\sum_{s_j} p(s_j|\vec{e}_X) V_j(s_j|\pi)$). For example, in our experiments of distributed task allocation, if agents X frequently interact with agent j but they are not well coordinated, then the value of $g(X, j)$ tends to be a large negative value (all expected rewards are negative). Here

ill-coordination means that agents X frequently generate events that cause agent j to be in states with low expected rewards. For instance, they send tasks to agent j when it is overloaded.

Obviously, if agents X do not effectively interact with agent j, then $g(X, j|\pi) = 0$ (because $E_X^j = \emptyset$). Now let us consider a special type of interactions among agents, called mutually exclusive interactions.

Definition 15. *Two nonempty disjoint agent sets X and Y are said to* **mutually exclusively interact** *with agent j, iff $E_X^j = \emptyset \vee E_Y^j = \emptyset \vee p(s_j^{t+1} = s_j, e_j^t = e_j, \vec{e}_X^t = \vec{e}_X, \vec{e}_Y^t = \vec{e}_Y) = 0$, for all $s_j \in S_j, e_j \in S_j \times A_j, \vec{e}_X \in E_X^j, \vec{e}_Y \in E_Y^j$.*

If X and Y mutually exclusively interact with agent j, then no two effective interactions generated by X and Y, respectively, will simultaneously occur to affect the state transition of agent j. In many applications [24, 131, 130], agents have such a type of interactions. For example, in network routing [24], the state space is defined by the destination of packages and each decision of an agent is triggered by one routing packet sent by one agent, so any two agents mutually exclusively interact with any third agent. Mutually exclusive interaction has the following property.

Proposition 1. *If X and Y mutually exclusively interact with agent j, then $g(X \cup Y, j|\pi) = g(X, j|\pi) + g(Y, j|\pi)$.*

Proof. Let E_X and E_Y be all events generated by X and Y, respectively.

$$
\begin{aligned}
g(X \cup Y, j|\pi) &= \sum_{\vec{e}_{XY} \in E^j_{X \cup Y}} p(s_j, \vec{e}_{XY}|\pi) V_j(s_j|\pi) \\
&= \sum_{\vec{e}_X \in E^j_X} \sum_{\vec{e}_Y \in E_Y} \sum_{s_j} p(s_j, \vec{e}_X, \vec{e}_Y|\pi) V_j(s_j|\pi) \\
&\quad + \sum_{\vec{e}_X \in E_X} \sum_{\vec{e}_Y \in E^j_Y} \sum_{s_j} p(s_j, \vec{e}_X, \vec{e}_Y|\pi) V_j(s_j|\pi) \\
&\quad - \sum_{\vec{e}^j_X \in E_X} \sum_{\vec{e}_Y \in E^j_Y} \sum_{s_j} p(s_j, \vec{e}_X, \vec{e}_Y|\pi) V_j(s_j|\pi) \\
&= \sum_{\vec{e}_X \in E^j_X} \sum_{s_j} p(s_j, \vec{e}_X|\pi) V_j(s_j|\pi) \\
&\quad + \sum_{\vec{e}_Y \in E^j_Y} \sum_{s_j} p(s_j, \vec{e}_Y|\pi) V_j(s_j|\pi) \\
&= g(X, j|\pi) + g(Y, j|\pi)
\end{aligned}
$$

□

Let $\mathcal{X}$ be all agents in a system and $\mathcal{X}_j \subseteq \mathcal{X}$ be a set of agents that effectively interact with agent j.

Proposition 2. *If every two agents in $\mathcal{X}_j$ mutually exclusively interact with agent j, then*

$$\rho_j(\pi_j|\pi_{-j}) = \sum_{x \in \mathcal{X}_j} g(\{x\}, j|\pi).$$

Proof.

$$
\begin{aligned}
\rho_j(\pi_j|\pi_{-j}) &= \sum_{s_j} p(s_j|\pi) V_j(s_j|\pi)|\pi) \\
&= \sum_{\vec{e}_X \in E^j_{\mathcal{X}_j}} p(\vec{e}_X|\pi) \sum_{s_j} p(s_j|\vec{e}_X) V_j(s_j|\pi) \\
&= g(\mathcal{X}_j, j|\pi) \\
&= \sum_{x \in \mathcal{X}_j} g(\{x\}, j|\pi)
\end{aligned}
$$

□

Corollary 1. *If every pair of agents in $\mathcal{X}$ mutually exclusively interact with any third agent, then*

$$\sum_{j \in \mathcal{X}} \sum_{x \in \mathcal{X}} w_j g(\{x\}, j|\pi) = \rho(\pi).$$

This corollary follows immediately from Lemma 8 and Proposition 2. Proposition 2 and Corollary 1 show how interactions contribute to the local and global performance, respectively, that is, the greater the absolute value of the weighted gain of interactions between two agents, the greater the (positive or negative) potential impact of their interactions on both the local and global performance. Although the properties of the gain of interactions we have just shown are valid in a restricted case, it can also be shown that the global performance measure can be tightly bounded by a weighted sum of gains of interactions among agents, which are approximately mutually exclusive. Therefore, the weighted gain can generally reflect the strength of interactions between agents, which is the basis of our self-organization approach.

7.2.2 Distributed Agent Clustering through Negotiation

Our clustering algorithm is intended to form a nearly decomposable organization structure, where interactions between clusters are generally weaker than interactions within clusters, to facilitate coordinating DRL. We assume all reward weights are equal and use the absolute value of the gain of interactions to measure the strength of interactions among agents. Supervisory organizations formed by using this measure will favorably generate rules and suggestions to improve ill-coordinated interactions (i.e. with a large negative gain) and maintain well-coordinated interactions (i.e., with a large positive gain), which potentially improve the performance of DRL. Our algorithm does not require interactions between agents to be mutually exclusive.

Due to bounded computational and communication resources, we limit the cluster size to control the quality and complexity of coordination. Our clustering problem is formulated

as follows: given a set of agents $\mathcal{X}$ and the maximum cluster size θ, subdivide $\mathcal{X}$ into a set of clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, such that

1. $\forall i = 1, \ldots, m, |C_i| \leq \theta$,
2. $\cup C_i = \mathcal{X}$ and $\forall i \neq j, C_i \cap C_j = \emptyset$,
3. The total utility of clusters $U(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} U(C_i)$ is maximal, where $U(C_i)$ is the utility of a cluster C_i defined as follows:

$$U(C_i) = \sum_{x_i, x_j \in C_i \text{ and } x_i \neq x_j} |g(\{x_i\}, x_j)| \tag{7.7}$$

Note that the total utility $U(\mathcal{C})$ has no direct relation to the system performance measure $\rho(\pi)$. The purpose of our clustering algorithm is not to directly improve the system performance, but to form proper supervisory organizations for coordinating learners that are ill-coordinated so as to potentially improve the learning performance.

Our clustering approach is distributed and based on an iterative negotiation process that involves a two roles: a buyer and a seller. A *buyer* is a supervisor who plans to expand its control and recruit additional agents into its cluster. A *seller* is a supervisor who has agents that the buyer would like to have. Supervisors can be buyers and sellers simultaneously. A *transaction* is to transfer a nonempty subset of boundary subordinates from a seller's cluster to a buyer's cluster. The *local marginal utility* is the difference between a cluster's utility before a transaction and the utility after the transaction. The *social marginal utility* is the sum of the local marginal utilities of both the buyer and the seller.

Based on these terms, our clustering problem can be translated into deciding which sellers the buyers should attempt to get agents from and which buyers the sellers should sell their agents to so that $U(\mathcal{C})$ is maximized.

The input to our clustering algorithm is an initial supervisory organization and the gain of interactions between agents. Figure 7.1 shows the dynamics of the negotiation protocol.

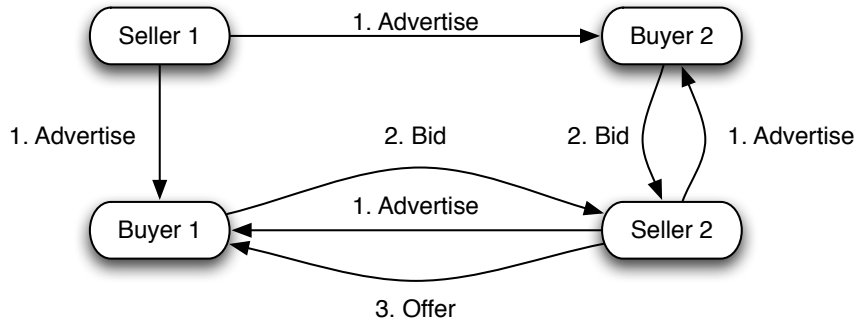


Figure 7.1. Self-organization negotiation protocol

Each supervisor only negotiates with its immediate supervisors. As our system is cooperative, our negotiation decisions are based on marginal social utility calculation. A round of negotiation consists of the following sub-stages:

1. Seller advertising: the supervisor of each cluster C_i sends an advertisement to each neighboring buyer. The advertisement contains local marginal utility $U^{lm}(C_i/X) = U(C_i) - U(C_i/X)$ of giving up each nonempty subset X of its subordinates adjacent to the buyer's cluster.

2. Buyer bidding: the supervisor of each cluster C_j waits for a period of time, collecting advertisements from neighboring supervisors. When the period is over, it calculates local marginal utility $U^{lm}(C_j \cup X) = U(C_j \cup X) - U(C_j)$ and then social marginal utility $U^{sm}(C_j, C_i, X) = U^{lm}(C_j \cup X) - U^{lm}(C_i/X)$ for introducing each nonempty subset X of subordinates of a seller of cluster C_i. If $U^{sm}(C_j, C_i, X)$ is the greatest social marginal utility and $U^{sm}(C_j, C_i, X) > 0$, then the buyer sends a bid to the supervisor of cluster C_i with the social marginal utility $U^{sm}(C_j, C_i, X)$; otherwise, do nothing.

3. Selling: given the multiple responses from buyers during a period time, the supervisor of cluster C_i chooses to transfer a subset of subordinates X to the cluster C_j if

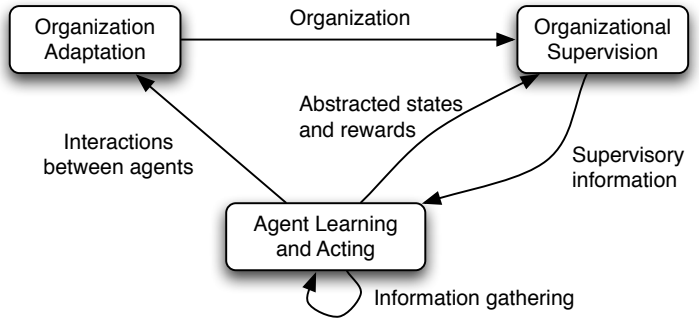


Figure 7.2. Extended supervision framework

$U^{sm}(C_j, C_i, X)$ is the maximal social marginal utility that the seller receives during this round.

The basic idea of our approach is similar to the LID-JESP algorithm [70] and the distributed task allocation algorithm in [48]. LID-JESP is used to generate offline policies for agents in a special DEC-POMDP, called ND-POMDP. However, we focus on agent clustering. Our negotiation strategy is also similar to that in [91], but uses one less sub-stage in each round of negotiation.

Proposition 3. *When our clustering algorithm is applied, the total utility $U(\mathcal{C})$ strictly increases until local optimum is reached.*

Sketch. By construction, only non-neighboring supervisors can transfer some subordinates to their neighboring clusters and they will only do this if the social marginal utility is positive, which results in an increase of the total utility $U(\mathcal{C})$. In addition, a supervisor's transferring subordinates to a neighboring cluster will not affect the utility of other neighboring clusters and non-neighboring clusters. Thus with each cycle the total utility is strictly increasing until local optimum is reached. □

7.2.3 Extended Supervision Framework

The gain of interactions is defined on the transition function, the reward function, and a specific joint policy. However, as all agents are learning their decision policies, interac-

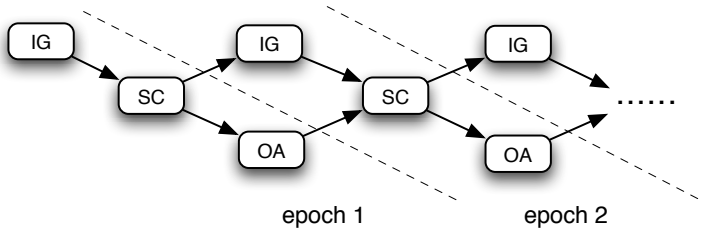


Figure 7.3. Iterations of three activities: information gathering (IG), supervisory control (SC), and organization adaptation (OA)

tions between agents may change over the time. To deal with this issue, we decompose the system runtime into a sequence of epochs. The gain of interactions between agents is approximately estimated from their execution trace during an epoch. Each epoch contains three activities: *information gathering*, and *supervisory control* and *organization adaptation*. The supervision framework proposed in [123] is now extended to allow dynamically evolving supervisory organizations for better coordinating DRL when agents are concurrently learning their decision policies. As shown in Figure 7.2, the extended framework contains these three interacting activities. Three activities iterate in the way as shown in Figure 7.3 during the whole system runtime.

Both information gathering activity and supervisory control activity have been discussed in detail in the previous chapter. With this extended framework, during the information gathering phase, each agent collects information about interactions from its neighbors, in addition to its execution sequence and reward information. After a period of time, agents will move to supervisory control phase, at the beginning of which each agent will calculate the gain of interactions with its neighbors and report it along with other information (i.e., abstracted states and rewards) to its supervisor. To avoid interfering the DRL supervision, organization adaption only happens after the supervisory control phase. However, since there is no communication between learning agents and their supervisors during the information gathering stage, organization adaption can be conducted concurrently with the next phase of information gathering. During this phase, using information of subordinates' in-

teractions with their neighbors, supervisors run our negotiation-based clustering algorithm and supervisor selection strategy to dynamically adapt the current supervisory organization. The resulting organization will be used for the next supervisory control activity. Initially, the system starts with a very simple supervisory organization, where each agent is its own supervisor. Then the supervisory organization is periodically evolving as agents are learning and acting.

7.3 Experiments

We evaluated our approach in a distributed task allocation problem (DTAP) [123] with Poisson task arrival distribution and exponentially distributed service time. Agents are organized in a network. Each agent may receive tasks from either the environment or its neighbors. At each time unit, an agent makes a decision for each task received during this time unit whether to execute the task locally or send it to a neighbor for processing. A task to be executed locally will be added to the local queue. Agents interact via communication messages and communication delay between two agents is proportional to the distance between them. The main goal of DTAP is to minimize the average total service time (ATST) of all tasks, including routing time, queuing time, and execution time.

7.3.1 Experimental Setup

We chose one representative MARL algorithm, the Weighted Policy Learner (WPL) algorithm [1], for each worker to learn task allocation policies. WPL is a gradient ascent algorithm where the gradient is weighted by $\pi(s, a)$ if it is negative; otherwise, it will weighted by $(1 - \pi(s, a))$. A worker's state is defined by a tuple $\langle l, f \rangle$, where l is the current work load (or total work units) in the local queue and f is a boolean flag indicating whether there is a task to be made a decision. Each neighbor corresponds to an action which forwards a task to that neighbor, and an agent itself corresponds to the action that put a task to the local queue. The reward $r(s, a)$ of doing an action a for an task is the

negative value of the expected service time to complete the task after doing a in state s, which is estimated from previous finished tasks. All agents use WPL with learning rate 0.001.

The abstracted state of a worker is projected from its states and defined by its average work load over a period of time τ ($\tau = 500$ in our experiments). The abstracted state of a supervisor is defined by the average load of its cluster, which can be computed from the abstracted states of its subordinates. A subordinate sends a report, which contains its abstracted state, to its supervisor every τ time period. Supervisors use simple heuristics to generate rules and suggestions. With an abstracted state $\langle \bar{l} \rangle$, a supervisor generates a rule that specifies, for all states whose work load exceeds $\bar{l}$, a worker should not add a new task to the local queue. This rule helps balance load within the cluster. A supervisor also generates positive (or negative) suggestions for its subordinates to encourage (or discourage) them forwarding more tasks to a neighboring cluster that has a lower (or higher) average load. The suggestion degree for each subordinate depends on the difference between the average load of two clusters, the number of agents on the boundary, and the distance of the subordinate to the boundary. Therefore, suggestions are used to help balance the load across clusters. The implementation detail of generating supervisory information is discussed in [122]. Our experiments use the receptivity function $\eta(s) = 1000/(1000+visits(s))$, where $visits(s)$ is the number of visits on state s.

To allow its supervisor to run our negotiation-based self-organization algorithm, each agent calculates the gain of interactions from other agents. As mentioned in Section 7.2.3, because of learning, each agent needs to approximately estimate each component in the definition of the gain of interactions from the history of its local executions and interactions with other agents in order to calculate it. In DTAP, one agent only interacts with its neighbors by forwarding tasks to them and its state does not affect states of its neighbors. Let $\vec{e}_k^j$ be the event of agent k, forwarding a task to agent j, that effectively interacts with agent j. To calculate $g(\{k\}, j|\pi)$, agent j estimates $p(\vec{e}_k^j|\pi)$ as the ratio of the number of

tasks received from agent k to the total number of received tasks and $p(s_j|\vec{e}_k^j)$ as the ratio of the number of visits on state s_j resulting from $\vec{e}_k^j$ to the total number of visits on this state, and uses its current learned policy π_j and reward function r_j.

Three measurements are evaluated: average total service time (ATST), average number of messages (AMSG) per task, time of convergence (TOC), and average cluster size (ACS). ATST indicates the overall system performance. AMSG takes into account all messages for routing task, coordination, and self-organization negotiation. To calculate TOC, we take sequential ATST values with certain size. If the ratio of those values' deviation to their mean is less than a threshold (we use threshold of 0.025), we consider the system stable. TOC is the start time of the selected points. ACS is the average cluster size in the system at TOC.

Experiments were conducted using a 18x18 grid network with 324 agents. All agents have the same execution rate and tasks are not decomposable. The mean of task service time is $\mu = 10$. We tested two patterns of task arrival:

Side Load where agents in a 3x3 grid at the middle of each side receive tasks with rate $\lambda = 0.8$ and other agents receive no tasks from the external environment.

Corner Load where only agents in the 8x8 grid at the upper left corner receive tasks from the external environment. Within that grid, the 36 agents at the upper left corner has the task arrival rate $\lambda = 0.25$ and the rest agents has the rate $\lambda = 0.7$.

We compared the DRL performance under four cases: *None, Fixed-Small, Fixed-Large, and Self-Org*. In the *None* case, no supervision is used to coordinate DRL. Both *Fixed-Small* and *Fixed-Large* cases use a fixed organization, the former with 36 clusters, each of which is a 3x3 grid, and the latter with 9 clusters, each of which is a 6x6 grid. The *Self-Org* case uses our self-organization approach to dynamically evolving supervision organization.

In each simulation run, ATST and AMSG are computed every 500 time units to measure the progress of the system performance. Results are then averaged over 10 simulation runs and the variance is computed across the runs.

7.3.2 Experimental Results

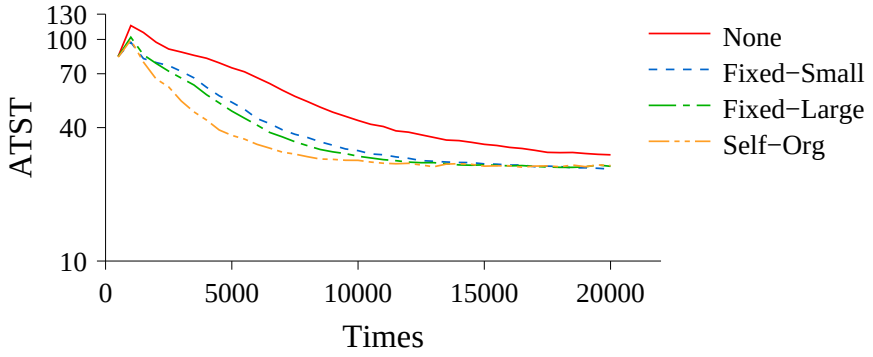


Figure 7.4. ATST under side load

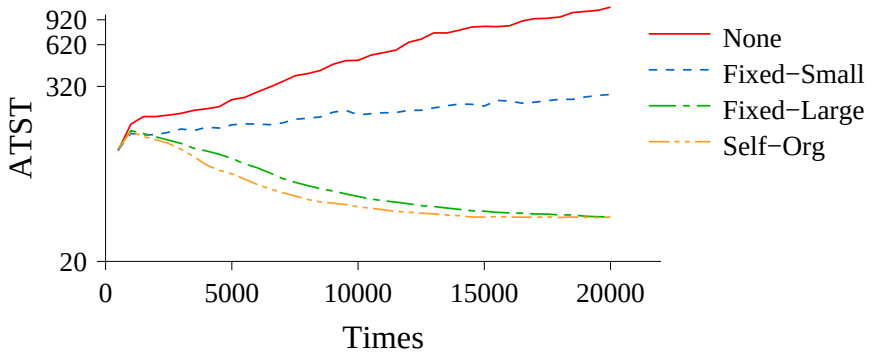


Figure 7.5. ATST under corner load

Figure 7.4 and 7.5 plot the trends of ATST, as agents learn, for different organization structures with different task arrival patterns. Note that the y axis in the plots is logarithmic. The supervision framework generally improves both the likelihood and speed of the learning convergence. Supervision with self-organized structure has a better learning curve than

that with predefined organization structures. This is because our self-organization approach clusters highly interdependent agents together, and focused coordination on them tends to greatly improve the system performance. The *Fixed-Small* case has a small cluster size and consequently some highly interdependent agents are not coordinated well. In contrast, the *Fixed-Large* case has a large cluster size, which enlarges both the view and control of each supervisor and potentially improve the system performance. However, with a large cluster size, an abstracted state of a cluster (generated by a supervisor) tends to lose detailed information about its subordinates, and also weakily interdependent agents are mixed with highly interdependent agents, both of which degrade the coordination quality.

Under corner load, the system with both *None* and *Fixed-Small* cases seems not to converge. For the *None* case, due to communication delay and limited views, agents in the top-left conner do not learn quickly enough knowledge about where light-loaded agents are. As a result, more and more tasks loop and reside in the top-left 8x8 grid. This makes the system load severely unbalanced and the system capability not well utilized, which causes the system load to monotonically increase. For the *Fixed-Small* case, because of a small cluster size, a supervisor's local view of the system may not be consistent with the global view. Some supervisors of overloaded clusters find their neighbors having even higher loads and consider their own clusters are "lightly" loaded. As a result, they generate incorrect directives for their subordinates, which degrade their normal learning.

Structure	ATST	AMSG	TOC	ACS
None	33.47 ± 1.67	5.81 ± 0.07	13000	0
Fixed-Small	29.09 ± 1.27	6.04 ± 0.11	10000	9
Fixed-Large	29.30 ± 1.46	6.16 ± 0.14	8500	36
Reorg	28.98 ± 1.15	6.59 ± 0.08	6500	14.50 ± 0.55

Table 7.1. Performance under side load

Table 7.1 and 7.2 show different measures for each supervision structure at their respective convergence time points. Due to the system divergence, both the *None* and *Fixed-Small* cases have no data under corner load. In addition to improving the convergence rate, the

Structure	ATST	AMSG	TOC	ACS
None	N/A	N/A	N/A	0
Fixed-Small	N/A	N/A	N/A	9
Fixed-Large	44.94 ± 2.10	11.26 ± 0.10	12500	36
Self-Org	42.87 ± 2.06	11.41 ± 0.05	10500	25.33 ± 2.16

Table 7.2. Performance under corner load

supervision framework also decreases the system ATST. Self-organization further improves the coordination performance, as indicated by its ATST and TOC. Because of negotiations, the self-organization case has a slightly heavier communication overhead than those of fixed organizations.

7.4 Summary

In this chapter, we address an important aspect of our supervisory framework to allow supervisory organizations to automatically evolve for better dynamically coordinating MARL. We formally define and analyze a type of interactions, called joint-event-driven interactions, among agents in a DEC-MDP. Based on this analysis, we develop a distributed self-organization approach that dynamically adapts hierarchical supervision organizations for coordinating DRL during the learning process. Experimental results demonstrate that dynamically evolving hierarchical organizations outperform predefined organizations in terms of both the probability and the quality of convergence. In the next chapter, we will deal with another important aspect of our supervisory framework: automating the supervision process for coordinating MARL without domain knowledge.

CHAPTER 8

AUTOMATING COORDINATION FOR MULTI-AGENT LEARNING IN ND-POMDPS

In Chapter 6, we present a general supervision framework for addressing challenges of scaling MARL to large complex MAS applications. Previous chapter addresses one important aspect of our supervision framework to allow supervisory organizations to automatically evolve for better dynamically coordinating MARL. Another important aspect of our supervision framework is how to coordinate MARL with little or no domain knowledge. As will be presented in this chapter, our attempt in this research direction focuses on a class of cooperative multi-agent decision making problems, which can be modeled by Networked Distributed POMDPs (ND-POMDPs) (a restricted version of DEC-POMDP). We tailor our supervision framework for coordinating MARL in ND-POMDP problems: making supervisors learn policies for their own subordinates and employing distributed constraint optimization (DCOP) techniques to automatically coordinate supervisors' learning without employing domain knowledge or heuristics. The work of this chapter was published in AAAI 2011 [125].

8.1 Introduction

Decentralized partially observable MDP (DEC-POMDP) provides a powerful framework for modeling cooperative multi-agent decision making problems under uncertainty. Due to the intractability of optimally solving general DEC-POMDPs, research has focused on restricted versions of DEC-POMDP that are easier to solve yet rich enough to represent many practical applications. Networked Distributed POMDP (ND-POMDP) [114] is

one such model that is inspired by a real-world sensor network coordination problem [53]. ND-POMDP assumes transition and observation independence and locality of interaction.

A rich portfolio of algorithms have been developed for solving ND-POMDPs [114, 61, 49]. One good feature of these techniques is that, although computing policies is centralized or requires extensive communication, executing computed policies does not require explicit communication. However, this feature may prevent agents from better coordination during execution when communication is allowed. In fact, in many practical applications, communications (at least between neighboring agents) are necessary for agents to perform tasks. For example, for target tracking in sensor networks, agents need to fuse their observations and actions to determine sensing results. The work [104] introduced communication in ND-POMDPs to periodically synchronize the belief state and extended existing algorithms to obtain policies with longer horizons. However, extensive communication is required for global synchronization, which is not scalable. In addition, all these algorithms for ND-POMDPs are offline techniques and require accurate models of the environment, which are usually costly to obtain in practice.

In this chapter, we present a model-free, scalable learning approach to developing policies for ND-POMDPs. Our approach synthesizes multi-agent reinforcement learning (MARL) and distributed constraint optimization (DCOP). By exploiting locality of interactions in ND-POMDPs, our approach factors a global joint action-value function and distributes the learning of the joint policy, which potentially scales up the learning to large-scale ND-POMDPs. Using communication between neighboring agents, our approach employs DCOP techniques to coordinate distributed learning to ensure the global performance. Our previous chapter presents a general supervisory framework for coordinating MARL, but did not provide a general coordination algorithm without exploiting domain knowldge. In this chapter, we demonstrate that DCOP algorithms can be used as general techniques for coordinating MARL in ND-POMDPs.

Coordinated reinforcement learning based on coordination graphs [30] has been explored in [31, 46] for factored MDPs. In contrast to these previous work, in this chapter, we explore coordinated multi-agent reinforcement learning in a principled way in ND-POMDPs and prove that our coordinated learning approach can learn the globally optimal policy for ND-POMDPs with a property, called *groupwise observability*. In addition, we also demonstrate that a max-sum algorithm [97] can be used for an approximate solution to our distributed coordination problem in learning, which requires limited communication overhead (typically scaling linearly with the number of agents) and computation. This DCOP algorithm can be readily implemented as an anytime algorithm to trade off solution quality and cost of computation and communication. Unlike the message-passing algorithm in [46], this algorithm can be directly used for coordinating interactions involving more than two agents. Experimental results show that, even in ND-POMDPs without groupwise observability, our approach scales to larger domains and performs significantly better and with orders of magnitude time savings (in the offline mode) over the previous best offline algorithm. Note that, as our approach needs communication during execution, a direct comparison among approaches is not appropriate. However, the offline results do provide a way to evaluate our approach by providing a baseline (i.e., nearly-optimal performance without communication).

8.2 Background

This section briefly introduces an illustrative problem in the sensor network domain, the ND-POMDP model, and basic learning approaches.

8.2.1 Illustrative Domain

This illustrative problem is motivated by a real-world challenge, where a network of agents (sensors) are used to track targets. Figure 8.1 shows a specific problem instance consisting of four sensors. Here, each sensor node can scan in one of four directions:

Figure 8.1. A 4-chain sensor configuration

North, South, East or West. To track a target and obtain the associated reward, two sensors with overlapping scanning areas must coordinate by scanning the same area simultaneously. For example, sensor1 needs to scan East and sensor2 needs to scan West simultaneously to track a target in location1. Thus, sensors have to act in a coordinated fashion. The movement of targets is unaffected by sensor agents. Sensors have imperfect observability of the target, so there can be false positive and negative observations. Sensors receive a reward on successfully tracking a target, and they incur a cost, when they either scan an area in an uncoordinated fashion or when the target is absent.

8.2.2 Networked Distributed POMDPs

Observe that sensors in this domain are mostly independent. Their state transitions, given the target location and the observations, are independent of the actions of the other agents. The only dependence arises from the fact that two agents must coordinate by scanning the same region to track a target. This dependence can be translated into a joint reward function. Such dependence is usually localized among a few agents (only two agents in this sensor network problem). The ND-POMDP model [114] was introduced to express such a type of interactions.

Definition 16. *An* **ND-POMDP** *is defined by the tuple* $\langle I, S, A, \Omega, P, O, R, b \rangle$*, where*

$I = \{1, \ldots, n\}$ *is a set of agent indices.*

$S = \times_{i \in I} S_i \times S_u$. S_i *refers to the local state of agent* i. S_u *refers to a set of uncontrollable states that are independent of the actions of the agents. In the sensor network*

example, S_i is empty, while S_u corresponds to the set of locations where targets can be present.

A $= \times_{i \in I} A_i$*, where A_i is the set of actions for agent i. For the sensor network example, $A_1 = \{N, W, E, S, Off\}$.*

$\Omega = \times_{i \in I} \Omega_i$ *is the joint observation set.*

P $P(s'|s,a) = P_u(s'_u|s_u) \cdot \prod_{i \in I} P_i(s'_i|s_i, s_u, a_i)$*, where $a = \langle a_1, \ldots, a_n \rangle$ is the joint action performed in joint state $s = \langle s_u, s_1, \ldots, s_n \rangle$ resulting in joint state $s' = \langle s'_u, s'_1, \ldots, s'_n \rangle$. (This models the transition independence.)*

O $O(\omega|s,a) = \prod_{i \in I} O_i(\omega_i|s_i, s_u, a_i)$*, where s is the joint state resulting after taking joint action a and receiving joint observation ω. (This models the observation independence.)*

R $R(s,a) = \sum_l R_l(s_l, s_u, a_l)$*. The reward function is decomposable among sub groups of agents referred by l. If $k = |l|$ agents $i_1, \ldots, i_k$ are involved in a particular sub group l, then s_l denotes the state of group l, i.e., $\langle s_{l1}, \ldots, s_{lk} \rangle$. Similarly, $a_l = \langle a_{l1}, \ldots, a_{lk} \rangle$. In the sensor domain, the reward function is expressed as the sum of rewards between sensor agents that have overlapping areas $(k = 2)$ and the reward functions for an individual agent's cost for sensing$(k = 1)$. Based on the reward function, an interaction hypergraph $G = (I, E)$ can be constructed, where I is a vertex (i.e., agent) set and E is a set of hyperlinks. A hyperlink $l \in E$ connects the subset of agents which form the reward component R_l. Note that this interaction hypergraph will be used to develop our learning approach in later sections.*

b $b = (b_u, b_1, \ldots, b_n)$ *is the initial belief for joint state $s = \langle s_u, s_1, \ldots, s_n \rangle \in S$ and $b(s) = b(s_u) \cdot \prod_{i \in I} b_i(s_i)$, where b_u and b_i are the initial distribution over S_u and S_i.*

The goal for ND-POMDPs is to compute a joint policy π that maximizes the total expected reward of all agents over a finite horizon T starting from b. Without communication,

agents can only act based on its local observations. In this case, a joint policy π is defined by $\langle \pi_1, \ldots, \pi_n \rangle$, where π_i refers to the individual policy of agent i that maps its history of observations to an action $a_i \in A_i$. If communication is allowed, a joint policy π can also be defined by one policy, called *global policy*, that maps from a history of joint observations to a joint action $a \in A$. This is because agents can exchange their observations and select actions based on joint observations. Obviously, the optimal global policy inherently performs better than the optimal set of individual policies. In this chapter, we assume agents can communicate (at least with their neighbors) during the execution time and focus on representing and learning the optimal *global policy* in a scalable way.

8.2.3 Basic Learning Approaches

To learn the joint policy, we need to define Q-function (or Q-value function). Let Q-function $Q(\vec{h}, a)$ represent the expected reward of doing joint action a with history $\vec{h}$ of joint observations and actions and behaving optimally from then on. The globally joint policy π can be derived from $Q(\vec{h}, a)$ by setting $\pi(\vec{h}) = \text{argmax}_{a \in A} Q(\vec{h}, a)$.

In principle, we can directly estimate $Q(\vec{h}, a)$ by using standard single-agent Q-learning:

$$Q(\vec{h}^t, a^t) = (1 - \alpha) Q(\vec{h}^t, a^t) + \alpha [r^t + \gamma \max_a Q(\vec{h}^{t+1}, a)] \tag{8.1}$$

where $\alpha \in (0, 1)$ is the learning rate, r^t is the immediate reward of doing a^t for observation history $\vec{h}^t$, $\gamma \in [0, 1]$ is the discount factor, which is usually set to 1 for a finite horizon. We call this approach *globally joint learning*. Although this approach leads to an optimal policy, it is practically intractable, because the policy space is exponential in the number of agents and the agents might not have access to the needed information (i.e., observations, actions, and rewards of all other agents) for learning and selecting actions.

At the other extreme, we can have the *independent learning* approach [27] in which agents ignore the actions and rewards of the other agents, and concurrently learn their own action-value functions solely based on their local observations and rewards. To provide lo-

cal rewards in ND-POMDPs, we can split the reward component R_l evenly among agents in group l. This approach is distributed, results in big storage and computational savings in the policy space, and does not require communication during learning and execution. However, this approach lacks coordination and might lead to oscillations or converge to local optimal policies. For example, in Figure 8.1, if location1, location2, and location3 always have targets with sensing reward 50, 60, and 50, respectively, then, by using independent learning approach, sensor2 and sensor3 will learn to always sense location2, which is locally optimal with average expected reward 60. However, the optimal policy is that sensor1 and sensor2 always sense location1 and sensor3 and sensor4 always sense location3, whose global expected reward is 100. Therefore, some form of coordination is needed in order to learn the globally optimal policy.

8.3 Coordinated Multi-Agent Reinforcement Learning

As discussed in the previous section, directly learning the globally joint policy in a centralized way is infeasible from a practical perspective, while independent learning is a distributed, scalable approach, but may yield poor global performance. In this section, we present a coordinated multi-agent learning approach for ND-POMDPs that attempts to achieve both scalability and optimality (or near-optimality). This approach distributes the learning by exploiting structured interactions in ND-POMDPs and coordinates distributed learning to ensure the global performance.

Our approach optimizes a decomposable Q-function $\hat{Q}(\vec{h}, a)$ that is used to approximate the global Q-function $Q(\vec{h}, a)$. This Q-function $\hat{Q}(\vec{h}, a)$ is defined as a sum of smaller local Q-functions based on hyperlinks in the interaction hypergraph of ND-POMDPs, that is,

$$\hat{Q}(\vec{h}, a) = \sum_{l \in E} Q_l(\vec{h}_l, a_l), \tag{8.2}$$

where $Q_l(\vec{h}_l, a_l)$ is the expected reward for agents on hyperlink l by doing joint action a_l^t at joint history $\vec{h}_l^t$ and behaving *globally* optimally from then on in respect to maximizing $\hat{Q}(\vec{h}, a)$. We will show in the next subsection that this approximation becomes exact for ND-POMDPs with a property called *groupwise observability*, which will lead to the theoretical result of optimality for our approach. In fact, this approximation is reasonable for general ND-POMDPs. This is because the global reward in ND-POMDPs is the sum of local rewards of groups defined on hyperlinks in the interaction hypergraph, and, as a result, $Q(\vec{h}, a)$ and $\hat{Q}(\vec{h}, a)$ are strongly positively correlated. Therefore, maximizing $\hat{Q}(\vec{h}, a)$ can potentially optimize $Q(\vec{h}, a)$. Our experimental results will verify this hypothesis on ND-POMDPs without the groupwise observability property.

Q-learning is used to learn the optimal $\hat{Q}(\vec{h}, a)$. With the decomposition in (8.2), the global Q-learning update rule in (8.1) can be rewritten as

$$\textstyle\sum_{l\in E} Q_l(\vec{h}_l^t, a_l^t) = \;\; (1-\alpha)\sum_{l\in E} Q_l(\vec{h}_l^t, a_l^t) + \alpha[\sum_{l\in E} r_l^t + \gamma \max_a \hat{Q}(\vec{h}^{t+1}, a)] \quad (8.3)$$

Note that the discounted future reward, $\max_a \hat{Q}(\vec{h}^{t+1}, a)$, can not be directly written as the sum of local discounted future rewards, because it depends on the joint action that maximizes the global value. Fortunately, we can accomplish this by defining the joint action $a^* = \text{argmax}_a \hat{Q}(\vec{h}^{t+1}, a)$ and $\max_a \hat{Q}(\vec{h}^{t+1}, a) = \hat{Q}(\vec{h}^{t+1}, a^*) = \sum_{l\in E} Q_l(\vec{h}_l^{t+1}, a_l^*)$. We are now able to decompose all terms in (8.3) and write the update rule for each group l:

$$Q_l(\vec{h}_l^t, a_l^t) = (1-\alpha) Q_l(\vec{h}_l^t, a_l^t) + \alpha[r_l^t + \gamma Q_l(\vec{h}_l^{t+1}, a_l^*)] \quad (8.4)$$

Similar to Sparse Cooperative Q-Learning [46], update rule in (8.4) is based on local reward and Q-function, except for a_l^*. Note that the local contribution $Q_l(\vec{h}_l^{t+1}, a_l^*)$ of group l to the global action value might be lower than $\max_{a_l} Q_l(\vec{h}_l^{t+1}, a_l)$, the maximizing value of its local Q-function, because it is unaware of the dependencies among groups.

We will use distributed constraint optimization (DCOP) techniques to compute a_l^*, which will be discussed later. Update rule in (8.4) is different from coordinated reinforcement learning approach in [31], where local Q-function update depends on the global reward signal and the global Q-value, which are not usually specifically tailored to local behaviors, thus resulting in slower learning convergence.

Using update rule in (8.4), our approach distributes the learning of the global function $\hat{Q}$ among groups. Our approach assumes that each group has a delegate agent (which can be chosen arbitrarily from a group) that learns Q_l on behalf of the group. The basic learning process is as follows. During each learning cycle t, after executing actions a_l^t, agents in group l receive and transmit their observations to the delegate agent of their group and the delegate agent receives its group reward signal r_l^t. Using its updated observation history $\vec{h}_l^{t+1}$, the delegate agent then computes the next best action a_l* for $\vec{h}_l^{t+1}$ by using a DCOP technique and updates its Q-function Q_l using rule (8.4). Finally, it distributes the next actions to its group members, which will be a_l* or some exploration actions.

The learned global Q-function is distributedly represented by local Q-functions of delegate agents. As a result, during execution, agents' action selections are computed online in a distributed manner by a DCOP algorithm from local Q-fuctions. Note that local Q-function $Q_l(\vec{h}_l^t, a_l^t)$ is defined on the observation history of group l, which scales exponentially with the horizon. To deal with a large horizon, one approach is to use a fixed-size window of observations, as we did in our experiments. Other more sophisticated approaches (i.e., utile suffix memory [65]) for dealing with partial observability can also be used with our approach.

In next two subsections, we will formally analyze the optimality of our approach and discuss how to compute joint action selections for learning or execution.

8.3.1 Optimality Analysis

In this section, we first define a property for ND-POMDPs, called *groupwise observability*, and then prove that our approach can learn an optimal policy for ND-POMDPs with this property.

Definition 17. *An ND-POMDP is said to have* **groupwise observability** *if, for all $l \in E$, the set of observations $\omega_l = \langle \omega_{l1}, \ldots, \omega_{lk} \rangle$ made by agents on hyperlink l together fully determine the current uncontrolled state, that is, if $\forall l \forall \omega_l \exists s_u : \Pr(s_u|\omega_l) = 1$.*

Note that this property does not imply that agents can observe their local states or states of other agents. It does imply that, for each agent $i \in l$, $P_i(s'_i|s_i, s_u, a_i, \omega_l) = P_i(s'_i|s_i, a_i, \omega_l)$ and $O_i(\omega_i|s_i, s_u, a_i, \omega_l) = O_i(\omega_i|s_i, a_i, \omega_l)$, which means, given joint observation ω_l, observation and transition of agent i on l are completely independent of observations and actions of other agents, and, as a result, its local belief update only depends on its local action and observation. This further implies that, in ND-POMDPs with groupwise observability, the local belief of agent $i \in l$ can be fully determined by its initial local state and the history of joint observations and actions of agents on l.

The theoretical result of optimality of our approach is as follows.

Theorem 3. *For ND-POMDPs with groupwise observability, under basic assumption of Q-learning and by using update rule (8.4), $Q_l(\vec{h}_l, a_l)$ will converge to the optimal $Q^*_l(\vec{h}_l, a_l)$, for all $l \in E$, and the policy $\pi^*(\vec{h}) = \text{argmax}_a \sum_{l \in E} Q^*_l(\vec{h}_l, a_l)$ is globally optimal.*

The proof for this theorem can be conducted by showing that Q-function $\hat{Q}$ defined in Equation (8.2) is exactly the same as the objective function Q of ND-POMDPs. This is because, if the approximation of $\hat{Q}$ is exact, then our coordinated learning approach described above is essentially a distributed version of update rule (8.1) that uses Q-learning, which leads to the global optimal $Q^*(\vec{h}, a)$. The exactness of this approximation for ND-POMDPs with groupwise observability will be shown by Proposition 5.

Our proof first defines a Q-function with state variables, then shows it is decomposable, and finally uses this result to prove the approximation of $\hat{Q}$ to Q is exact for ND-POMDPs with groupwise observability.To simplify the equations, we introduce some abbreviations:

$$
\begin{aligned}
p_i^t &\equiv P_i(s_i^{t+1}|s_i^t, s_u^t, a_i^t) \cdot O_i(\omega_i^{t+1}|s_i^{t+1}, s_u^{t+1}, a_i^t) \\
p_u^t &\equiv P_u(s_u^{t+1}|s_u^t) \\
r_l^t &\equiv R_l(s_l, s_u, a_l) \\
Q^t &\equiv Q^t(s^t, \vec{h}^t, a^t) \\
Q^{t*} &\equiv \max_a Q^t(s^t, \vec{h}^t, a) \\
Q_l^t &\equiv Q_l^t(s_l^t, s_u^t, \vec{h_l}^t, a_l^t)
\end{aligned}
$$

The global Q-function $Q(s^t, \vec{h}^t, a^t)$ with state will satisfy the Bellman equation:

$$
Q(s^t, \vec{h}^t, a^t) = R(s^t, a^t) + \gamma \sum_{s^{t+1}, w^{t+1}} p_u^t p_1^t \dots p_n^t Q^{t*},
$$

where $\vec{h}^{t+1}$ is $\vec{h}^t$ appended by $\langle a^t, \omega^{t+1} \rangle$.

Let b^t be the belief state at time t. As b^t is fully determined by the initial belief b and history $\vec{h}^t$ of joint observations and actions, we have

$$
Q(\vec{h}^t, a^t) = \sum_{s \in S} b^t(s) Q(s^t, \vec{h}^t, a^t). \tag{8.5}
$$

Similarly, we define a Q-function for each hyperlink l:

$$
Q_l(s_l^t, s_u^t, \vec{h_l}^t, a^t) = r_l^t + \gamma \sum_{s_l^{t+1}, \omega_l^{t+1}} p_u^t p_{l1}^t \dots p_{lk}^t Q_l^{t+1*},
$$

where $\vec{h_l}^{t+1}$ is $\vec{h_l}^t$ appended by $\langle a_l^t, \omega_l^{t+1} \rangle$ and Q_l^{t+1*} denotes $Q_l(s_l^{t+1}, \vec{h_l}^{t+1}, a_l^*)$, where a_l^* is the *globally* optimal joint action taken by agents on l in the next global state and history of joint observations and actions of all agents.

For ND-POMDPs with groupwise observability, as $b_u^t(s_u)$ is fully determined by history $\vec{h_l}^t$ of joint observations and actions, and, for $i \in l$, $b_i^t(s_i)$ is fully determined by the initial belief $b_i(s_i)$ and history $\vec{h_l}^t$, we then have

$$Q(\vec{h_l}^t, a_l^t) = \sum_{s_l, s_u} b_l^t(s_u, s_l) Q_l(s_l, s_u, \vec{h_l}^t, a_l^t). \tag{8.6}$$

Proposition 4. *In ND-POMDPs, the global function $Q^t(s^t, \vec{h}^t, a^t)$ for any finite horizon T is decomposable, that is,*

$$Q^t(s^t, \vec{h}^t, a^t) = \sum_{l \in E} Q_l^t(s_l^t, s_u^t, \vec{h_l}^t, a_l^t). \tag{8.7}$$

Proof. Proof is by mathematical induction. Proposition holds for $t = T - 1$ because $r^t = \sum_{l \in E} r_l^t$ and there is no future reward. Assume it holds for t where $1 \leq t \leq T - 1$, that is, $Q^t = \sum_{l \in E} Q_l^t$.

Now let us show that proposition holds for $t - 1$.

$$\begin{aligned}
Q^{t-1} &= R(s^{t-1}, a^{t-1}) + \gamma \sum_{s^t, w^t} p_u^{t-1} p_1^{t-1} \dots p_n^{t-1} Q^{t*} \\
&= \sum_{l \in E} r_l^{t-1} + \gamma \sum_{s^t, w^t} p_u^{t-1} p_1^{t-1} \dots p_n^{t-1} \sum_{l \in E} Q_l^{t*} \\
&= \sum_{l \in E} [r_l^{t-1} + \gamma \sum_{s_l^t, s_u^t, w_l^t} p_u^{t-1} p_{11}^{t-1} \dots p_{lk}^{t-1} Q_l^{t*}] \\
&= \sum_{l \in E} Q_l^{t-1}
\end{aligned}$$

□

Based on Proposition 4, Equation 8.5 and 8.6, we can show an exact decomposition of the Q-function without state.

Proposition 5. *In ND-POMDPs with groupwise obserbability, the global Q-value function $Q^t(\vec{h}^t, a^t)$ for any finite horizon T is decomposable, that is,*

$$Q^t(\vec{h}^t, a^t) = \sum_{l \in E} Q_l^t(\vec{h_l}^t, a_l^t). \tag{8.8}$$

Proof.

$$\begin{aligned} Q(\vec{h}^t, a^t) = & \sum_{s_u, s_1, \ldots, s_n} b_u^t(s_u) b_1^t(s_1) \ldots b_n^t(s_n) \cdot \\ & \sum_{l \in E} Q_l^t(s_l, s_u, \vec{h_l}^t, a_l^t) \\ = & \sum_{l \in E} \sum_{s_l, s_u} b_l^t(s_u, s_l) Q_l^t(s_l, s_u, \vec{h_l}^t, a_l^t) \\ = & \sum_{l \in E} Q_l^t(\vec{h_l}^t, a_l^t). \end{aligned}$$

□

This proposition completes the proof of Theorem 3.

8.3.2 Optimal Joint Action Selection

Our learning approach requires computing the joint action that maximizes the global Q-value function for updating local Q-functions or for acting during execution. We can formulate this problem as a DCOP, which is defined by a set of discrete variables $a = \{a_1, \ldots, a_n\}$, where $a_i \in A_i$ is controlled by agent i and represents its action choice, and a set of functions $Q = \{Q_l | l \in E\}$, where Q_l is the Q-value function for hyperlink l. Note that history $\vec{h}$ is fixed for every computation, so we will ignore it in the following discussion and denote $Q_l(\vec{h}, a_l)$ by $Q_l(a_l)$. The goal is to find the joint action a^*, such that the global Q-value function, the sum of all Q-functions, is maximized, that is, $a^* = \text{argmax}_a \sum_{l \in E} Q_l(a_l)$. We can represent this DCOP as a factor graph by creating a node for

each variable and for each function and connecting a function node to a variable node if the corresponding function is dependent upon that variable. The resulting graph is bipartite.

A variable elimination algorithm [30] can be used to compute an optimal solution for this DCOP, but it requires extensive communication and computation (scaling exponentially with the induced width of the agent interaction graph). In this chapter, we investigate the max-sum algorithm [97] for an approximate solution, which requires much less communication and computation and can be readily implemented as an anytime algorithm to trade off the quality and efficiency of computing joint actions. Unlike the max-plus algorithm in [46], this algorithm can be directly used for coordinating interactions involving more than two agents.

The max-sum algorithm operates directly on the factor graph, and does so by specifying the messages that should be passed from variable to function nodes, and from function nodes to variable nodes, which are defined as follows:

- **Message from variable node i to function node l:**

$$q_{i\to l}(a_i) = \sum_{g\in\mathcal{F}_i\setminus l} r_{g\to i}(a_i) + c_{il}$$

where $\mathcal{F}_i$ is a vector of function indexes, indicating which function nodes are connected to variable node i, and c_{il} is a normalizing constant to prevent the messages from increasing endlessly in cyclic graphs.

- **Message from function node l to variable node i:**

$$r_{l\to i}(a_i) = \max_{\mathbf{a}_l\setminus a_i}[Q_l(\mathbf{a}_l) + \sum_{g\in\mathcal{V}_l\setminus i} q_{g\to l}(a_g)]$$

where $\mathcal{V}_l$ is a vector of variable indexes, indicating which variable nodes are connected to function node l and $\mathbf{a}_l\setminus a_i = \{a_g : g \in \mathcal{V}_l\setminus i\}$.

Here variable node i is agent i who needs to select its action and function node l is the delegate agent of hyperlink l that hosts the Q-value function Q_l. If the factor graph is cycle-free, the algorithm is guaranteed to converge to the optimal global solution such that each agent i can find its optimal action a_i^* by locally calculating $a_i^* = \text{argmax}_{a_i} z_i(a_i)$, where $z_i(a_i) = \sum_{g \in \mathcal{F}_i} r_{g \to i}(a_i)$. Otherwise, there is no guarantee of convergence. However, extensive empirical results show that, even in this case, the algorithm frequently provides good solutions. Before convergence, the value $z_i(a_i)$ of agent i calculated from incoming messages is actually an approximation of the exact value of action a_i given other agents act optimally. Therefore, the max-sum algorithm can be implemented as an anytime algorithm by controlling the number of rounds of passing messages, which will trade off the quality and efficiency (or communication cost) of the action selection. In addition, the max-sum algorithm is essentially distributed. Its messages are small (linearly scaling with the maximum number of actions of agents), the number of messages typically varies linearly with the number of agents and hyperlinks, and its computational complexity scales exponentially with the maximum size of hyperlinks (which typically is much less than the total number of agents).

8.4 Experiments

To evaluate our coordinated learning (CL) approach in general ND-POMDPs, we experimented it in the illustrative sensor network domain, which does not have the groupwise observability property. We compared CL with the independent learning (IL) approach (described in the Background Section) and CBDP [49], one of the most efficient algorithms for ND-POMDPs. We conducted experiments with configurations shown in Figure 8.2. The first three configurations are introduced in [61], but we changed their initial beliefs to an uniform distribution over ten states to increase problem difficulty. The 25-grid sensor network has two targets with the same sensing rewards as 15-3D, but has a larger state space and longer target paths.

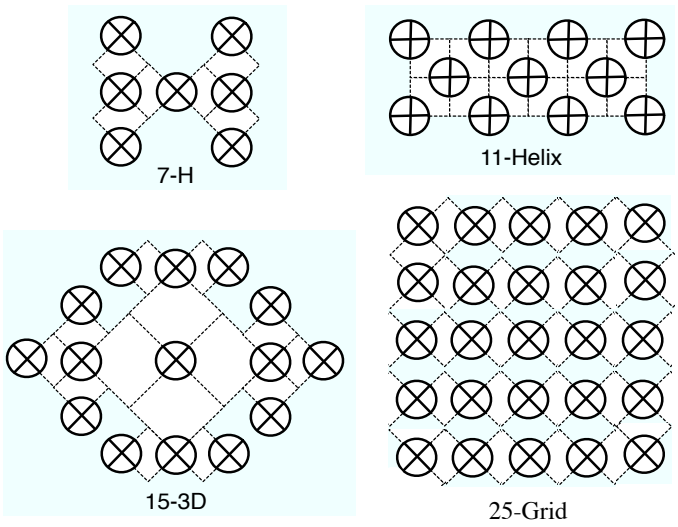


Figure 8.2. Sensor network configurations

Since both CL and IL are model-free, we develop a simulator for ND-POMDPs to learn and evaluate policies. The evaluation process is as follows: for each ND-POMDP, we use CBDP to solve it and get its joint policy, then run both learning approaches in a simulator for that ND-POMDP, whose learning time is set to some ratio of CBDP's computation time, and, finally evaluate learned policies and CBDP's policy in the simulator. The solution quality for each horizon is indicated by the expected global reward for that horizon. Solution quality is computed over 10000 simulation runs. Results are then averaged over 10 experiments and the deviation is computed, which is very small (under 5) and not shown properly in the following figures. The learning rate α is set to 0.001 and discount factor $\gamma = 1$. Both learning approaches learned policies that map fixed-windows of observations (with size ≤ 4) to an action even for scanarios with horizon greater than 5. To trade off the speed and solution quality, we restricted the max-sum algorithm passing messages at most 4 rounds for each joint action computation (except for experiments of controlling communication).

Figure 8.3 (a) shows the solution quality of CL and IL with different learning time on the configuration 15-3D with horizon $T = 10$. The configuration 15-3D is the most

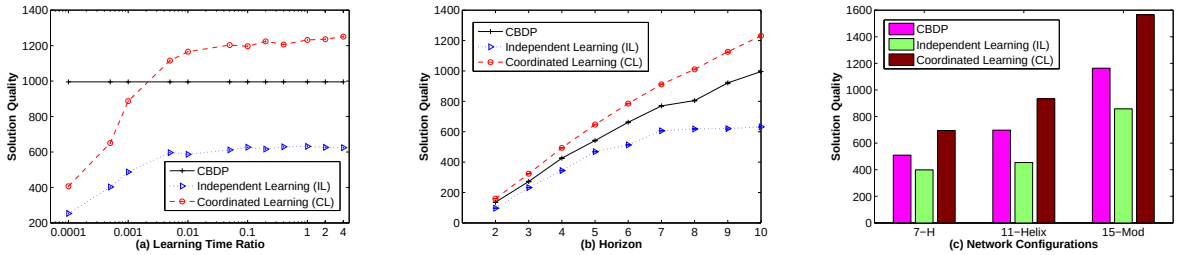


Figure 8.3. Solution quality over (a) different ratios of learning time of IL and CL to CBDP's policy computation time on 15-3D with horizon $T = 10$, (b) over different horizons on 15-3D, and (c) different network configurations with $T = 10$. Note that IL and CL in (b) and (c) use the same learning time as CBDP's policy computation time.

complex problem instance for CBDP. The x axis represents the ratio of learning time to CBDP's computation time, which is plotted with a logarithmic scale. The performance of both CL and IL generally increases with more training time. We observe that CL can learn policies, whose performance surpasses that of CBDP's policy, with learning time two orders of magnitude less than CBDP's computation time. However, IL performs much worse than CL and CBDP. One reason is that, as we have discussed, IL can only converge to local optima, which is far away from the global optimal solution on the configuration 15-3D. This result actually illustrates the importance of the coordination during learning and execution. Another reason is that IL (and CL) uses fixed-window policy that maps up to 4 observations to an action, while CBDP's policies with horizon $T = 10$ maps up to 9 observations to an action. We did observe that IL could perform comparably or better than CBDP on smaller problems with small horizon (e.g., one the domain 11-Helix with 5 horizon).

Figure 8.3 (b) shows the solution quality over a range of horizons on the configuration 15-3D. We can see that the solution quality of CL linearly increases with the horizon size, whose increase rate is greater than CBDP. This indicates that CL can potentially scale better than CBDP with the horizon size. Figure 8.3 (c) shows the solution quality on other

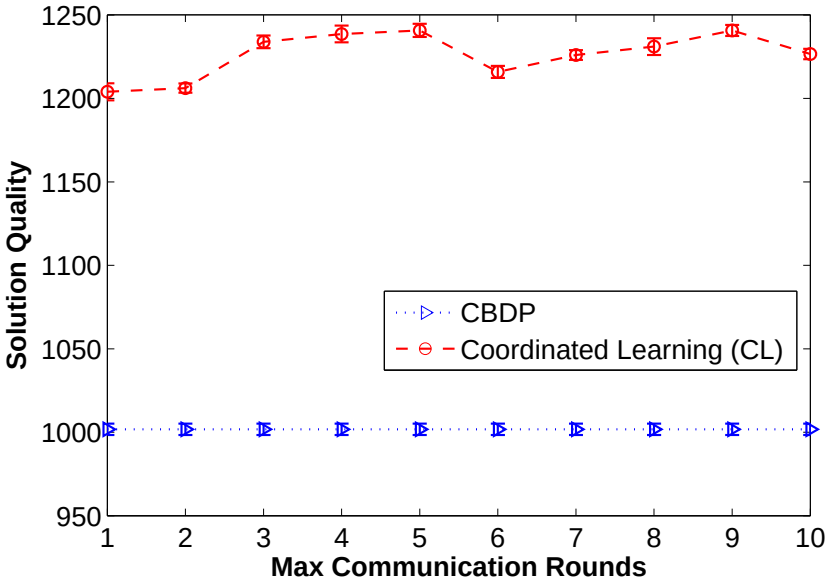


Figure 8.4. Trade-off of solution quality and communication

configurations, where 15-Mod is the modified version of 15-3D with different target paths. Consistent with results on 15-3D, CL performs best, then CBDP, and finally IL.

By controlling the maximum round of message passing between agents and their group delegates for computing joint actions, we can trade off solution quality and cost of communication and computation. Figure 8.4 show the solution quality of CL over different maximum rounds of message passing on the domain 15-3D with horizon 10 and the same learning time as CBDP's computation time. We can observe that CL still performs significantly better than CBDP, even when using only one-round message passing. Note that when using fixed learning time, more rounds of message passing do not necessarily yield better learning performance. This is because, although using more rounds of message passing computes better joint actions, it results in more communication and computation at each learning cycle and learning with less total cycles.

We also evaluated CL and IL on the 25-grid problem, where CBDP could not scale even to horizon 2. The learning time is set to 200 seconds for horizon 5 and linearly increases

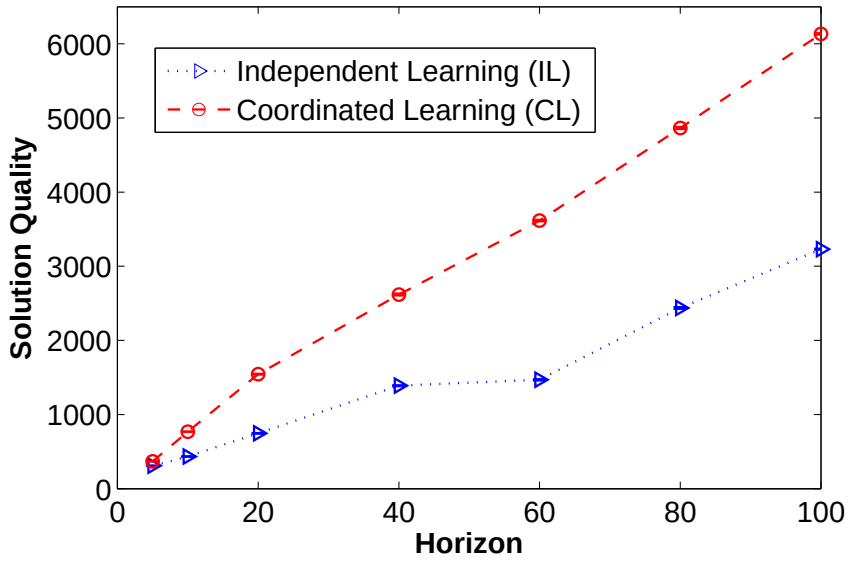


Figure 8.5. Solution quality for a range of horizons on 25-grid

with the horizon. Figure 8.5 shows solution quality over horizons up to 100. The solution quality of CL almost doubled that of IL and increases linearly with the horizon.

8.5 Summary

In this chapter, we demonstrate that DCOP algorithms can be used as general techniques for automatically coordinating MARL in ND-POMDPs without depending on domain knowledge. Our model-free learning approach for ND-POMDPs decomposes and distributes the learning of the optimal global joint policy by exploiting its structured interactions through a decomposable reward function and independence among agents. Distributed learning is coordinated through joint action selection computed by distributed constraint optimization (DCOP) techniques, which ensure the optimality of the learning for ND-POMDPs with *groupwise observability*. By exploiting the property of locality of interactions in ND-POMDPs, the learning complexity potentially scales linearly with the number of agents. To trade off solution quality and communication and computation efficiency, a max-sum algorithm is used to compute an approximate solution for our DCOP.

Experimental results show that, even in ND-POMDPs without groupwise observability, by exploiting extra communication during learning and execution, this approach significantly outperforms off-line construction of nearly-optimal no-communication policies.

Part IV

CONCLUSIONS AND FUTURE WORK

CHAPTER 9

SUMMARY AND CONTRIBUTIONS

Multi-agent systems (MAS) are increasingly being advocated for use in building robust adaptive complex systems. Uncertainty and complexity are inherent in most real-world MAS applications, where the environment characteristics are usually unknown and tens to thousands of agents interact with limited communication bandwidth and limited observability. In this thesis, we has primarily focused on addressing a central challenge in MAS research: *how to design coordination policies for autonomous agents that act in such uncertain, complex environments to optimize global performance*? We have developed a new multi-agent learning paradigm that allows agents to concurrently learn to effectively coordinate in large domains. To scale up the learning, this paradigm exploits locality of interaction and non-local information in a coherent way. Using this paradigm, agents concurrently learn their policies based on local observations, and, meanwhile, their learning processes are coordinated by a non-local control mechanism to ensure the global learning performance. In this thesis, we has developed both efficient algorithms for multi-agent learning (MAL) with limited observability and a scalable control framework for coordinating MAL. We also have applied and evaluated this learning paradigm in diverse problem domains, including distributed task allocation, network routing, and sensor networks.

Multi-Agent Learning Techniques: In a MAS, as agents interact and concurrently learn their policies, the environment becomes non-stationary from the perspective of each individual agent. The convergence guarantee for single-agent reinforcement learning does not hold for multi-agent settings. We developed a gradient-based MAL algorithm by extending Q-learning and applied it to optimizing online distributed resource allocation in

Cloud Computing. Empirical results demonstrated an impressive performance of this algorithm in convergence and solution quality. However, this algorithm may not converge in competitive scenarios, which can occur in cooperative MAS when we design local reward signals (instead of using the single global reward) for agents' learning.

To consider both cooperative and non-cooperative scenarios, we introduced the concept of policy prediction and augmented the basic gradient-based learning algorithm to achieve two properties: best-response learning and convergence. These two properties were analyzed for a class of stochastic games under the assumption of full observability. Inspired by this analysis, we have developed a multi-agent reinforcement learning algorithm for more challenging scenarios where agents observe only the reward signal of choosing an action. Empirical results demonstrated that this algorithm outperformed state-of-the-art MAL techniques in term of convergence.

Coordinating Multi-Agent Learning: The theoretical convergence guarantee of existing MAL algorithms is still limited for simple stochastic games. By exploiting locality of interaction alone, MAL algorithms still converge slowly, converge to inferior equilibria, or even diverge in large-scale complex settings. We developed a distributed supervisory control framework that has the potential to address these issues with scaling MAL. This framework exploits non-local information and multi-level organizational control to coordinate and guide the agents' learning process. It introduces a more global view into the learning process of individual agents without incurring significant overhead and exploding their policy spaces; it coordinates the learning behaviors of tightly coupled agents by constraining their learning processes while still leaving agents to react autonomously to local reward signals. The organizational structure can dynamically evolve by a self-organization approach as the learning progresses in order to more effectively coordinate the learning agents. This coordination results in both speeding up and increasing the likelihood of convergence by reducing the occurrence of oscillatory behavior among agents learning with limited observability in a non-stationary environment and focusing agents' exploration.

Additionally, it also results in improved overall solution quality due to coordination directives that are based on a more global view of current learning.

In addition, We also attempted to address one challenge of this framework: automatically coordinating MARL. We tailored our supervision framework to a class of multi-agent decision making problems, called networked distributed POMDPs (ND-POMDPs) so that distributed constraint optimization techniques can be used to automatically coordinate multi-agent learning without employing domain knowledge or heuristics. Its convergence and optimality were proved for a restricted class of ND-POMDPs. By using a message-passing algorithm, this approach can be implemented in a distributed way as an anytime algorithm that trades off solution quality and cost of communication and computation.

CHAPTER 10

FUTURE DIRECTIONS AND OPEN PROBLEMS

There are five main aspects to extend our existing research: 1) developing theoretical models (e.g., agent interaction model) to understand the applicability of the supervision framework and the concurrent evolution of a supervisory organization and agents' decision policies; 2) designing general techniques to allow automated supervision with little or no domain knowledge; 3) expanding supervision modes by allowing supervisors to shape rewards for learners, provide non-local state information to learners, and control learning parameters (e.g., learning rate) of agents; 4) developing techniques for a new form of transfer learning to allow agents to dynamically share learned knowledge with other agents that are concurrently learning in the network; and 5) evaluating the paradigm performance in more realistic, complicated environments.

We begin discussion of the different research directions by introducing a model for agent interaction. As will be discussed, we feel this model will inform many of the proposed future research directions. Although our agent interaction model only captures the instantaneous state of the learning process and not the dynamics of the learning process or the supervision process, it is still a foundational step to understand and improve the effectiveness of the supervisory control for improving MARL. This interaction model represents a generalization from our previous work [126] in its ability to capture more complex interaction patterns among agents than what was possible in our earlier work.

10.1 Theoretical Models and Analysis

The complexity of multi-agent systems arises from the interactions among agents. The analysis of interactions is a key to understanding how a MARL algorithm performs in MAS. We have developed an interaction model that defines and quantifies interactions among agents and analyzes how interactions are related to the system performance. The analysis of our current interaction model is restricted to a special type of interaction, called completely mutually exclusive interactions, where no two agents concurrently interact with a third agent. *One of future research directions is to extend this model to a general setting, which we believe will not only be the basis for dynamically evolving supervisory organizations [126], but also a key to developing automated supervision techniques with little or no domain knowledge, and understanding why and when our supervision framework performs effectively.*

10.1.1 Extended Interaction Model

Our quantified agent interaction model builds on a factored DEC-MDP [32] model that represents the multi-agent sequential decision-making problem in a collaborative MAS. Our interaction model is different from the work by Allen [6], which quantifies interactions between agents only based on the problem definition and without taking account of agents' policies.

Definition 18. *An n-agent factored DEC-MDP is defined by a tuple $\langle S, A, T, \mathcal{R} \rangle$, where*

- $S = S_1 \times \cdots \times S_n$ *is a finite set of world states, where S_i is the state space of agent i*
- $A = A_1 \times \cdots \times A_n$ *is a finite set of joint actions, where A_i is the action set for agent i*
- $T : S \times A \times S \rightarrow \Re$ *is the transition function. $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the probability of transitioning to the next state $\mathbf{s}'$ after a joint action $\mathbf{a} \in A$ is taken by agents in state $\mathbf{s}$*

- $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ *is a set of reward functions.* $R_i : S \times A \rightarrow \Re$ *provides agent* i *with an individual reward* $\mathbf{r_i} \in R_i(\mathbf{s}, \mathbf{a})$ *for taking action* $\mathbf{a}$ *in state* $\mathbf{s}$*. The global reward is the sum of all local rewards:* $R(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} R_i(\mathbf{s}, \mathbf{a})$

A (joint) policy $\pi : S \times A \rightarrow \Re$ is a function which returns the probability of taking (joint) action $\mathbf{a} \in A$ for any given state $\mathbf{s} \in S$. The goal is to derive an optimal policy that maximizes the average expected reward per time step, which measures the system performance. MARL is used by agents to learn efficient approximate policies in a factored DEC-MDP environment, especially when the transition and reward function is unknown. Each agent learns its local policy based on its local observation and the reward in presence of other agents, who are also learning a policy under the same conditions. The local policy $\pi_i : S_i \times A_i \rightarrow \Re$ for agent i returns the probability of taking action $a_i \in A_i$ in local state $s_i \in S_i$. The objective of agent i is to learn an optimal policy π_i^* to maximize its own average expected reward per time step.

Our interaction model characterizes a type of interaction among agents, called *joint-event-driven interactions*, in a DEC-MDP model.

Definition 19. *A* **primitive event** $e_j = \langle s_j, a_j \rangle$ *generated by agent* j *is a tuple that includes a state and an action on that state. A* **joint event** $\vec{e}_X = \langle e_{j_1}, e_{j_2}, \ldots, e_{j_h} \rangle$ *contains a set of primitive events generated by agents* $X = \{j_1, j_2, \ldots, j_h\}$*. A joint event* $\vec{e}_X$ *occurs iff all of its primitive events occur.*

For brevity, events discussed in this paper refer to joint events. Our definition of a joint event is different from that of an event in [13], where an event occurs if any one of its primitive events occurs. An event is used to capture the fact that some agents did some specific activities. A primitive event can be generated by either an agent or the external environment. For convenience, we treat the external environment as an agent.

Definition 20. *A* **joint-event-driven interaction** $i_j^X = \langle \vec{e}_X, e_j \rangle$ *from a set of agents* X *onto agent* j *is a tuple that includes a joint event* $\vec{e}_X$ *and a primitive event* e_j*. A joint-event-*

driven interaction i_j^X is **effective** *iff the event $\vec{e}_X$ affects the distribution over the resulting state of event e_j, that is, $\exists s_j \in S_j$ such that $p(s_j^{t+1} = s_j | e_j^t = e_j) \neq p(s_j^{t+1} = s_j | e_j^t = e_j, \vec{e}_X^t = \vec{e}_X)$, where t is the time.*

We define a measure for the strength of interactions among agents. Let $E_X^j = \{\vec{e}_X | \exists e_j \in S_j \times A_j$ such that interaction $\langle \vec{e}_X, e_j \rangle$ is effective$\}$ be all joint events generated by a set of agents X that effectively interact with agent j. Let $V_j(s_j|\pi) = \sum_{a_j} \pi_j(s_j, a_j) \mathbf{E}[r_j(s_j, a_j)|\pi]$ be the expected value of being in state s_j, where π_j is the policy of agent j, and $\mathbf{E}[r_j(s_j, a_j)|\pi]$ is the expected reward of executing action a_j in state s_j.

Definition 21. *The* **value of interactions** *from a set of agents X to agent j, given a joint policy π, is*

$$vi(X, j|\pi) = \sum_{\vec{e}_X \in E_X^j} p(\vec{e}_X|\pi) \sum_{s_j} p(s_j|\vec{e}_X, \pi) V_j(s_j|\pi),$$

where $p(\vec{e}_X|\pi)$ is the probability that event $\vec{e}_X$ occurs and $p(s_j|\vec{e}_X)$ is the probability of being in state s_j after $\vec{e}_X$ occurs.

The value of interactions will be used to analyze the effectiveness of our supervision framework. As mentioned earlier, it is also used in the supervisory organization adaptation for deciding which agents should be jointly supervised. The value of interactions is affected by two factors: how frequently agents effectively interact (reflecting on $p(\vec{e}_X|\pi)$) and how well they are coordinated (reflecting on $\sum_{s_j} p(s_j|\vec{e}_X) V_j(s_j|\pi)$). For example, in our experiments of distributed task allocation, if agents X frequently interact with agent j but they are not well coordinated, then the value of $g(X, j)$ tends to be a large negative value (all expected rewards are negative). Here ill-coordination means that agents X frequently generate events that cause agent j to be in states with low expected rewards. For instance, in a distributed task allocation problem, they send tasks to agent j when it is overloaded.

Definition 22. *Two nonempty disjoint agent sets X and Y are said to ϵ-***mutually-exclusively interact** *with agent j iff $E_X^j = \emptyset \vee E_Y^j = \emptyset \vee \sum_{\vec{e}_X \in E_X^j} \sum_{\vec{e}_Y \in E_Y^j} p(s_j^{t+1} = s_j, \vec{e}_X^t = \vec{e}_X, \vec{e}_Y^t =$

$\vec{e}_Y) \leq (1-\epsilon) \cdot \min(\sum_{\vec{e}_X \in E_X^j} p(s_j^{t+1} = s_j, \vec{e}_X^t = \vec{e}_X), \sum_{\vec{e}_Y \in E_Y^j} p(s_j^{t+1} = s_j, \vec{e}_Y^t = \vec{e}_Y))$, *for all* $s_j \in S_j$, *where* $0 \leq \epsilon \leq 1$.

This introduction of ϵ-mutually-exclusive interaction represents a significant extension of our early work.

10.1.2 Effectiveness Analysis

If X and Y ϵ-mutually-exclusively interact with $\epsilon = 1$, which is called completely mutually exclusive interactions, with agent j, then no two effective interactions generated by X and Y, respectively, will simultaneously occur to affect the state transition of agent j. We can prove that if any two agents completely mutually exclusively interact with a third agent in a factored DEC-MDP, then the sum of values of all possible interactions from one agent to another is equal to the measure of the system performance [126]. This result reveals how interactions contribute to the global performance for this case where $\epsilon = 1$.

This summation relationship between values of interactions among agents and the system performance measure explains why our supervision framework performs effectively to improve the speed, quality, and likelihood of the learning convergence in experiments of our preliminary work. For example, in our experiments of the distributed task allocation problem (DTAP), every two agents completely mutually exclusively interact with a third agent. Based on the current learning state of agents, the heuristic we used generates supervisory information that guides agents to explore the state-action space, which reduces the frequency of interactions with large negative values. This supervision information also increases the frequency of interactions with small negative values. However, since the function between the frequency of interactions and the value of interactions in DTAP is convex, the overall value of interactions among agents still increases. Therefore, our supervision framework coordinates and guides agents to learn policies that lead to the better system performance. Even if agents may be able to learn such policies on their own eventually, using our supervision framework significantly speeds up the learning process.

Future research can develop theoretical results for general cases with arbitrary ϵ. We speculate that values of interactions among agents have a nearly summation relationship to the global performance measure for large ϵ, but not for small ϵ. If this is the case, our supervision framework, with heuristics for generating supervisory information that manages the frequency of interactions, will perform effectively in problems with large ϵ where the function between the frequency and value of interactions is convex (which normally holds for various problems). Then interesting questions arise: what supervisory information will perform effectively in problems with small ϵ? How can supervisors automatically generate such supervisory information online?

10.1.3 Self-Organization Analysis

As described in Section 4.3, we developed a self-organization approach to dynamically evolving the supervisory organization that is coordinating and guiding agents' learning. Our self-organization approach is based on our interaction model developed above. Our approach is intended to form a nearly decomposable organization structure, where interactions between clusters are generally weaker than interactions within clusters, to facilitate coordinating MARL. We use the absolute value of the value of interactions to measure the strength of interactions among agents. Supervisory organizations formed by using this measure will favorably generate rules and suggestions to improve ill-coordinated interactions (i.e., with a large negative value) and maintain well-coordinated interactions (i.e., with a large positive value). Experimental results showed that a dynamically evolving supervisory organization can better speed up the learning process than predefined, static organizations. Empirical results can be plausibly explained by observations in human organizations that a nearly decomposable organization structure can improve the coordination quality and reduce the coordination complexity. Future research can develop formal results by using our interaction model to understand why and when a dynamically evolving supervisory organization performs better.

Through experiments, we found that our supervision framework with a dynamically evolving supervisory organization performed better than that with a fixed "learned" supervisory organization. A "learned" organization is obtained by running our supervision framework with the self-organization approach until the system converges, starting with a simple organization, called local supervisory organization, where each agent is its own supervisor and has a very limited view. We also found that our self-organization approach starting with a local supervisory organization performs better than that with a "learned" one where each supervisor has a larger cluster and a broader viewer of the system. Those empirical results plausibly imply that, to optimize its performance, our supervision framework may require different supervisory organizations at different learning stages of agents. At the early learning stage of agents, supervisory organizations with a small cluster (i.e., each supervisor has a very limited view) outperform those with a large cluster. We hypothesize that, at the early learning stage, agents' policies change very fast and state information gathered by supervisors with a larger cluster has more variance and is more unreliable. This unreliable state information of subordinates can lead to improper supervisory information, which may mislead subordinates' learning. As the self-organization approach is based on our interaction model, *we will work on formally understanding through the model why and when the concurrent evolution of the supervisory organization and decision policies of agents perform better than learning policies under a fixed, learned organization.*

10.2 Automating Supervision

In our earlier work, the supervisory control framework exploited domain knowledge to generate supervisory control information based on the current learning state of agents for coordinating agents' learning processes. Automating supervision reduces this dependency on domain knowledge and facilitates the application of the framework. Distributed constraint optimization (DCOP) techniques can be used to automatically coordinate distributed learning in some restricted class of problems. However, for certain circumstances,

such approaches may have increased communication overhead and policy search spaces. To improve the applicability of the proposed learning paradigm, It is highly desirable to develop general techniques for automating coordination with little or no domain knowledge. *Our research for this direction will be conducted in two phases. First, we plan to develop general heuristic-based techniques based on our interaction model to automate supervision. As heuristics usually do not perform effectively for all cases, we will then develop more general approaches, such as using learning algorithms to learn how to supervise agents' learning*.

As supervision is concerned with how to coordinate interactions among agents, we can use our interaction model as a basis to develop techniques to automate the generation of supervisory information. Our interaction model is based on the DEC-MDP model, which is domain-independent. The first technique we will explore is one in which each supervisor is trying to balance its cluster's interactions with its neighboring clusters by generating supervisory information that increases the frequency of interactions with large values, and decreases the frequency of interactions with small values. Based on the analysis in Section 10.1, when the function between the frequency and the value of interactions is convex (which holds for most practical problems) and interactions among agents are ϵ-mutually-exclusive with large ϵ, then this technique has a high probability of performing effectively to improve the quality or speed of the MARL convergence.

As heuristics-based techniques are usually simple and effective, but may not work well for some cases (e.g., where interactions among agents are ϵ-mutually-exclusive with small ϵ), we need to develop additional techniques. One direction we will pursue is to formalize each supervisor's decision making as a Markov Decision Process (MDP) and then solve it offline or learn its policy directly by using MARL algorithms. The challenge is to define the state space, action space, and reward function. The goal of each supervisor is to find $rules$ and $suggestions$ to its subordinates to maximize its local utility. The action space

is defined by the set of rules and suggestions and its reward is the aggregated reward of its subordinates.

A supervisor's decision making needs to take account of information from its subordinates and its neighboring clusters. It is more feasible and scalable for each supervisor to define its decision state with the *abstract state*, instead of real states, of subordinates and neighboring clusters. Each agent can demonstrate both fast and slow dynamics in how its features change. Fast dynamics of an agent are exhibited by the changes of such features as those that represent interactions with other agents, its local state, and its policy (or value function). Slow dynamics are exhibited by the changes of an agent's *abstract state*. The abstract state is defined by a vector of features, which can be projected from features with fast dynamics by using such techniques as:

- Using partial components of a feature and ignoring other components that do not affect slow dynamics
- Using statistics (e.g., mean, mode) of a feature generated over the temporal or spatial scale
- Replacing a fast-changing feature with its distribution parameters if its changes follow some statistical distribution

Similarly, each cluster also has fast and slow dynamics. Fast dynamics of a cluster are exhibited by that of its members. Slow dynamics of a cluster are captured by the changes of its supervisor's abstract state. The abstract state of a supervisor is projected either from the abstract states of its subordinates or directly from features with fast dynamics of its subordinates.

10.3 Other Supervision Modes

Our preliminary work laid out a general supervision framework for coordinating MARL. However, our current implementation provides only one way of supervising MARL (by

coordinating agents' exploration policies). *One of future research directions is to develop other ways of supervising MARL:*

- *Expanding the view of learning agents by providing non-local information*
- *Shaping rewards of learning agents*
- *Manipulating the learning parameters (e.g., the learning rate)*

Expanding agents' view by providing non-local information will relieve the burden of agents learning such information. Because of bounded communication and computation resources in practical, large-scale multi-agent systems, each agent interacts only with a limited number of agents, called neighbors, and, to be scalable, the learning of each agent has been restricted to using information received only from its neighbors. With our supervision framework, each supervisor obtains a broader view of the system with low communication overhead (since this information is only periodically provided), so it can directly provide non-local state information about non-neighboring agents and the system environment to its subordinates. How does a supervisor decide and compute what non-local information is useful for a particular subordinate? How do subordinates integrate non-local state information into their learning process? One potential way is to reshape or extend in a controlled fashion subordinates' state space that more accurately represents the world state. Is it possible to integrate non-local information into the local decision state without expanding the size of the state space?

In a single-agent setting, reward shaping [72] is a common approach to speeding up reinforcement learning by supplying an agent with additional reward signals to encourage some particular actions. Similar ideas may also be applicable for multi-agent settings. Within a supervisory organization, each supervisor can compute an aggregated reward of its cluster from information of its subordinates. This cluster reward reflects how well-coordinated the collective actions of agents in the cluster are, which is closer to the system performance measure than the local reward of any individual agent. However, it may not be a good idea to use the cluster reward as the local reward signal of subordinates, because

the cluster reward has only a weaker relationship with the state-action pair of an individual agent. One trade-off approach is to combine the cluster reward r_c and the local reward r_l, e.g., resulting in the new reward $r' = \eta * r_c + (1 - \eta) * r_l$, where $0 \leq \eta \leq 1$, as the learning signal for an individual agent. The parameter η can be adjusted dynamically based on the learning progress.

Almost all MARL algorithms have the learning rate parameter. Properly setting [16] and adapting the learning rate can improve both the speed and likelihood of the MARL convergence. For example, the infinitesimal gradient ascent (IGA) algorithm [92] does not converge to a Nash equilibrium in some multi-agent settings (e.g., repeated zero-sum games), but the GIGA-WoLF algorithm [23], which is an extension of IGA, has a better convergence property than IGA by properly varying the learning rate, although the convergence property is still restricted to a very limited setting. Although it is still not clear how each supervisor should manipulate the learning rate for its subordinates within a supervisory organization, we feel that its broader view of the system provides a basis for its decision.

10.4 Transfer Learning

The idea of *transfer learning* has recently been applied to reinforcement learning tasks to speed up the learning. In conventional transfer approaches [58, 95, 119, 105, 47, 106], the core idea of transfer learning is that experience gained in learning to perform one task, called a source task, can help improve learning performance in a related, but different, task, called a target task. To be effective, those approaches normally require thorough experience in the source task. *In multi-agent settings, our supervision framework provides opportunities to develop transfer learning techniques for agents to share their learned experience in order to speed up their learning, even when agents do not have full experiences with all states in their policy space*.

Through experiments, we found that, due to a slightly different initial environment, learning agents in a MAS may have very different exploration experiences, even when they have similar state-action space. For example, a group of agents A have a lot of experience in a set of states S_1, while a group of agents B have a lot of experience in a set of states S_2. However, because of a non-stationary environment (due to concurrent learning), after some period of time, group A begins to explore some states of S_2 and group B begins to explore some states of S_1. In such situations, sharing learning experience between A and B will reduce the time for exploration and speed up their learning process.

With our supervision framework, a supervisor can act as a demand-supply matching center for its subordinates. When a learning agent believes it has rich experience in some states, it may upload this learned knowledge to its supervisor. The learned knowledge can be represented by the value function or the policy. In contrast, when an agent begins to explore some new states, it may send a request to its supervisor to see if there is similar experience that has been gained by some other agents in the cluster. How does an agent know whether its experience in some states is rich enough? One simple measure is the number of visits in a state. Will this measure be good enough? If several agents offer their learned knowledge, how should they be combined together, e.g., using the most experienced one or their weighted average? When state-action spaces among agents are not the same, how is the mapping between agents defined or learned? To reduce communication overhead of a supervisor and its subordinates, we will also need to define projection functions to generate abstract states from real local states.

10.5 Performance Evaluation

We will evaluate the performance of our proposed framework using four domains to assess the broad impact of our work. The first domain is based on cooperative graphical games [43] and provides the minimum complexity needed to evaluate our framework. Additionally, three different realistic multi-agent testbeds will be used to verify the gener-

ality of our proposed frameworks: 1) cloud computing [129], 2) peer-to-peer information retrieval [130], and 3) wireless network routing [121].

Evaluating and analyzing the performance of an adaptive, self-organizing network of agents is challenging due to the large number of parameters at play. We will focus our study on the effects of following evaluation dimensions: the network structure (e.g., small-world, scale-free, random), the underlying learning algorithm (e.g., WPL [3], GIGA-WoLF [23]), the dynamics of the system (e.g., agents joining and leaving, change of incoming task patterns), the agent population (heterogeneity, distribution over the network), and the complexity of applications (e.g., varying the ϵ parameter of interaction patterns).

We intend to evaluate how our supervision framework improves the performance of MARL algorithms in terms of the quality, speed, and likelihood of the learning convergence. The overall system reward at the convergence time point indicates the convergence quality. The convergence speed is measured by the number of learning cycles to reach the convergence time point. We can define the core concept "convergence" at three different levels: the overall system reward averaged over time, the individual expected reward change averaged over time and agents, and the individual policy change averaged over time and agents. At the coarse level, defining on the overall system reward may hide the underlying agent dynamics. At the finest level, the convergence defining on individual policies is hard to reach because of the dynamic and open environment. We also evaluate the communication overhead incurred by our supervision framework in term of the number of messages used for the supervision after the learning converges.

To obtain intuitive explanations on the performance of our supervision framework, we plan to visualize the learning dynamics using different levels of detail. For initial verification, the individual policy evolution over time will be used, which is more suitable for small scale MAS. In large scale MAS we will investigate more aggregate measures, such as the policy entropy of individual agents. We will also use open-source network anal-

ysis software (e.g. Network Workbench [107]) to visualize and analyze the supervisory organization adaptation.

BIBLIOGRAPHY

[1] Abdallah, Sherief, and Lesser, Victor. Learning the task allocation game. In *AAMAS'06* (2006).

[2] Abdallah, Sherief, and Lesser, Victor. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS'07* (2007).

[3] Abdallah, Sherief, and Lesser, Victor. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research 33* (2008), 521–549.

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

[5] Agogino, Adrian K., and Tumer, Kagan. Quicr-learning for multi-agent coordination. In *In Proceedings of the 21st National Conference on Artificial Intelligence* (2006).

[6] Allen, Martin William. Interactions in decentralized environments. *Open Access Dissertations* (2009).

[7] Aron, Mohit, Druschel, Peter, and Zwaenepoel, Willy. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Measurement and Modeling of Computer Systems* (2000), pp. 90–101.

[8] Arpaci-dusseau, Andrea C., and Culler, David E. Extending proportional-share scheduling to a network of workstations. In *Proceedings of Parallel and Distributed Processing Techniques and Applications* (1997).

[9] Balch, Tucker. Learning roles: Behavioral diversity in robot teams. In *AAAI Workshop on Multiagent Learning* (1997).

[10] Balch, Tucker. Reward and diversity in multirobot foraging. In *IJCAI-99 Workshop on Agents Learning About, From and With other Agents* (1999).

[11] Banerjee, Bikramjit, and Peng, Jing. Generalized multiagent learning with performance bound. *Autonomous Agents and Multi-Agent Systems 15,* 3 (2007), 281–312.

[12] Barto, Andrew G., Sutton, Richard S., and Anderson, Charles W. Neuronlike adaptive elements that can solve difficult learning control problems. *Artificial neural networks: concept learning* (1990), 81–93.

[13] Becker, Raphen, Lesser, Victor, and Zilberstein, Shlomo. Decentralized Markov Decision Processes with Event-Driven Interactions. In *AAMAS'04* (2004), vol. 1, pp. 302–309.

[14] Bellman, Richard E. *Dynamic Programming*. Princeton University Press, 1957.

[15] Bernstein, Daniel S., Givan, Robert, Immerman, Neil, and Zilberstein, Shlomo. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research 27*, 4 (2002), 819–840.

[16] Bernstein, Daniel S., and Zilberstein, Shlomo. Reinforcement learning for weakly-coupled mdps and an application to planetary rover control. In *Proceedings of the Sixth European Conference on Planning* (Toledo, Spain, 2001).

[17] Bianchi, Reinaldo A. C., Ribeiro, Carlos H. C., and Costa, Anna H. R. Heuristic selection of actions in multiagent reinforcement learning. In *IJCAI'07* (Hyderabad, India, 2007).

[18] Bolch, Gunter, Greiner, Stefan, de Meer, Hermann, and Trivedi, Kishor S. *Queueing networks and markov chains*. Wiley, 1998.

[19] Borgers, Tilman, and Sarin, Rajiv. Learning through reinforcement and replicator dynamics. Else working papers, ESRC Centre on Economics Learning and Social Evolution, 1997.

[20] Boutilier, Craig. Planning, learning and coordination in multiagent decision processes. In *TARK '96: Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge* (San Francisco, CA, USA, 1996), Morgan Kaufmann Publishers Inc., pp. 195–210.

[21] Bowling, Michael. Convergence problems of general-sum multiagent reinforcement learning. In *IN PROCEEDINGS OF THE SEVENTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING* (2000), Morgan Kaufmann, pp. 89–94.

[22] Bowling, Michael. Convergence and no-regret in multiagent learning. In *NIPS'05* (2005), pp. 209–216.

[23] Bowling, Michael, and Veloso, Manuela. Multiagent learning using a variable learning rate. *Artificial Intelligence 136* (2002), 215–250.

[24] Boyan, Justin A., and Littman, Michael L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *NIPS'94* (1994), vol. 6, pp. 671–678.

[25] Brafman, Ronen I., and Tennenholtz, Moshe. Efficient learning equilibrium. In *Advances in Neural Information Processing Systems (NIPS-2002)* (2002).

[26] Chrisman, Lonnie. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *In Proceedings of the Tenth National Conference on Artificial Intelligence* (1992), AAAI Press, pp. 183–188.

[27] Claus, Caroline, and Boutilier, Craig. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI'98* (1998), AAAI Press, pp. 746–752.

[28] Conitzer, Vincent, and Sandholm, Tuomas. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning 67*, 1 (2007), 23–43.

[29] Greenwald, Amy, and Hall, Keith. Correlated-q learning. In *In AAAI Spring Symposium* (2003), AAAI Press, pp. 242–249.

[30] Guestrin, Carlos, Koller, Daphne, and Parr, Ronald. Multiagent planning with factored mdps. In *NIPS-14* (2001), pp. 1523–1530.

[31] Guestrin, Carlos, Lagoudakis, Michail G., and Parr, Ronald. Coordinated reinforcement learning. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning* (San Francisco, CA, USA, 2002), Morgan Kaufmann Publishers Inc., pp. 227–234.

[32] Guestrin, Carlos Ernesto. *Planning under uncertainty in complex structured environments*. PhD thesis, Stanford University, Stanford, CA, USA, 2003.

[33] han Chang, Yu, Ho, Tracey, and Kaelbling, Leslie Pack. All learning is local: Multi-agent learning in global reward games. In *Proceedings of Neural Information Processing Systems (NIPS-03)* (2003).

[34] Hansen, Eric A., Bernstein, Daniel S., and Zilberstein, Shlomo. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th national conference on Artifical intelligence* (2004), AAAI'04, AAAI Press, pp. 709–715.

[35] Horling, Bryan, and Lesser, Victor. Using quantitative models to search for appropriate organizational designs. *Autonomous Agents and Multi-Agent Systems 16*, 2 (2008), 95–149.

[36] Horling, Bryan, Mailler, Roger, and Lesser, Victor. A Case Study of Organizational Effects in a Distributed Sensor Network. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)* (Beijing, China, 2004), pp. 51–57.

[37] Howard, Ronald A. *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[38] Hu, J, and Wellman, M P. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning* (1998), pp. 242–250.

[39] Hu, Junling, and Wellman, Michael P. Nash q-learning for general-sum stochastic games. *JOURNAL OF MACHINE LEARNING RESEARCH 4* (2003), 1039–1069.

[40] Hu, Junling, and Wellman, Michael P. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research 4* (2003), 1039–1069.

[41] Jaakkola, Tommi, Jordan, Michael I., and Singh, Satinder P. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation 6*, 6 (1994), 1185–1201.

[42] Kaisers, Michael, and Tuyls, Karl. Faq-learning in matrix games: Demonstrating convergence near nash equilibria, and bifurcation of attractors in the battle of sexes. In *AAAI'11* (2011).

[43] Kakade, Sham, Kearns, Michael, Langford, John, and Ortiz, Luis. Correlated equilibria in graphical games. In *EC '03: Proceedings of the 4th ACM conference on Electronic commerce* (New York, NY, USA, 2003), ACM, pp. 42–47.

[44] Kapetanakis, Spiros, and Kudenko, Daniel. Improving on the reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Second Symposium on Adaptive Agents andMulti-agent Systems (AISB02)* (2002).

[45] Kapetanakis, Spiros, and Kudenko, Daniel. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI02)* (2002).

[46] Kok, Jelle R., and Vlassis, Nikos. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research 7* (2006), 1789–1828.

[47] Konidaris, George, and Barto, Andrew G. Building portable options: Skill transfer in reinforcement learning. In *IJCAI* (2007), pp. 895–900.

[48] Krainin, Michael, An, Bo, and Lesser, Victor. An Application of Automated Negotiation to Distributed Task Allocation. In *IAT'07* (2007), pp. 138–145.

[49] Kumar, Akshat, and Zilberstein, Shlomo. Constraint-based dynamic programming for decentralized pomdps with structured interactions. In *AAMAS* (2009).

[50] Kumar, Shailesh, and Miikkulainen, Risto. Confidence based dual reinforcement q-routing: An adaptive online network routing algorithm. In *IJCAI '99* (1999), pp. 758–763.

[51] Lauer, Martin, and Riedmiller, Martin. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning* (2000), Morgan Kaufmann, pp. 535–542.

[52] Leslie, David S., and Collins, E. J. Individual q-learning in normal form games. *SIAM Journal on Control and Optimization 44*, 2 (2005), 495–514.

[53] Lesser, V., Ortiz, C., and Tambe, M., Eds. *Distributed Sensor Networks: A Multiagent Perspective (Edited book)*, vol. 9. Kluwer Academic Publishers, 2003.

[54] Lin, Long-Ji, and Mitchell, Tom M. Memory approaches to reinforcement learning in non-markovian domains. Tech. rep., Tech. rep. CMU-CS-92-138, Carnegie Mellon University, School of Computer Science., 1992.

[55] Littman, Michael L. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning* (1994), pp. 157–163.

[56] Littman, Michael L. Friend-or-foe q-learning in general-sum games. In *ICML '01* (2001), pp. 322–328.

[57] Littman, Michael L. Value-function reinforcement learning in markov games. *Cognitive Systems Research 2*, 1 (2001), 55–66.

[58] Liu, Yaxin, and Stone, Peter. Value-function-based transfer for reinforcement learning using structure mapping. In *AAAI* (2006).

[59] Makar, Rajbala, Mahadevan, Sridhar, and Ghavamzadeh, Mohammad. Hierarchical multi-agent reinforcement learning. In *Autonomous Agents'01* (2001), pp. 246–253.

[60] Mangasarian, O. L., and Stone, H. Two-person nonzero-sum games and quadratic programming. *Journal of Mathematical Analysis and Applications 9*, 3 (1964), 348 – 355.

[61] Marecki, Janusz, Gupta, Tapana, Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Not all agents are equal: Scaling up distributed pomdps for agent networks. In *AAMAS* (2008), pp. 485–492.

[62] Mataric, Maja. Learning to behave socially. In *From Animals to Animats: International Conference on Simulation of Adaptive Behavior* (1994), MIT Press, pp. 453–462.

[63] Mataric, Maja J. Reinforcement learning in the multi-robot domain. *Autonomous Robots 4* (1997), 73–83.

[64] Mccallum, R. Andrew. Overcoming incomplete perception with utile distinction memory. In *In Proceedings of the Tenth International Conference on Machine Learning* (1993), Morgan Kaufmann, pp. 190–196.

[65] Mccallum, R. Andrew. Instance-based utile distinctions for reinforcement learning with hidden state. In *In Proceedings of the Twelfth International Conference on Machine Learning* (1995), Morgan Kaufmann, pp. 387–395.

[66] McKelvey, Richard D., and McLennan, Andrew. Computation of equilibria in finite games. In *HANDBOOK OF COMPUTATIONAL ECONOMICS* (1996), Elsevier, pp. 87–142.

[67] Melo, Francisco S., and Veloso, Manuela. Learning of coordination: exploiting sparse interactions in multiagent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Richland, SC, 2009), AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, pp. 773–780.

[68] Moore, Andrew W., and Atkeson, Christopher G. Prioritized sweeping: Reinforcement learning with less data and less time. In *Machine Learning* (1993), pp. 103–130.

[69] Nagayuki, Yasuo, Ishii, Shin, and Doya, Kenji. Multi-agent reinforcement learning: An approach based on the other agent's internal model. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-00)* (2000).

[70] Nair, Ranjit, Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Networked distributed pomdps: a synthesis of distributed constraint optimization and pomdps. In *AAAI'05* (2005), pp. 133–139.

[71] Nash, John. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences 36*, 1 (1950), 48–49.

[72] Ng, Andrew Y., Harada, Daishi, and Russell, Stuart. Policy invariance under reward transformations: theory and application to reward shaping. In *ICML'99* (1999), pp. 278–287.

[73] Osborne, Martin J., and Rubinstein, Ariel. *A course in game theory*. The MIT Press, 1994.

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

[75] Peng, Jing, and Williams, Ronald J. Incremental multi-step q-learning. *Mach. Learn. 22*, 1-3 (1996), 283–290.

[76] Perko, Lawrence. *Differential equations and dynamical systems*. Springer-Verlag Inc., 1991.

[77] Peshkin, Leonid, and Savova, Virginia. Reinforcement learning for adaptive routing. In *International Joint Conference on Neural Networks (IJCNN)* (2002).

[78] Petrik, Marek, and Zilberstein, Shlomo. Average-reward decentralized markov decision processes. In *IJCAI* (2007), pp. 1997–2002.

[79] Price, Bob, and Boutilier, Craig. Implicit imitation in multiagent reinforcement learning. In *ICML'99* (1999), pp. 325–334.

[80] Ramamritham, K., Stankovic, J. A., and Zhao, W. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Comput. 38*, 8 (1989), 1110–1123.

[81] Rosenstein, Michael T., and Barto, Andrew G. Supervised actor-critic reinforcement learning. In *Learning and Approximate Dynamic Programming: Scaling Up to the Real World* (2004), J. Si, A. Barto, W. Powell, and D. Wunsch, Eds., John Wiley and Sons, pp. 359–380.

[82] Sandholm, Tuomas, and Crites, Robert H. On multiagent q-learning in a semi-competitive domain. In *IJCAI '95: Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems* (London, UK, 1996), Springer-Verlag, pp. 191–205.

[83] Schaerf, Andrea, Shoham, Yoav, and Tennenholtz, Moshe. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research 2* (1995), 475–500.

[84] Schmidhuber, Jurgen. Realistic multi-agent reinforcement learning. In *Learning in Distributed Artificial Intelligence Systems. Working Notes of the 1996 ECAI Workshop* (1996).

[85] Schmidhuber, Jurgen, and Zhao, Jieyu. Multi-agent learning with the success-story algorithm. In *ECAI Workshop LDAIS* (1997), Springer, pp. 82–93.

[86] Sen, Sandip, Sen, Ip, Sekaran, Mahendra, and Hale, John. Learning to coordinate without sharing information. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence* (1994), pp. 426–431.

[87] Shapley, Lloyd Stowell. Stochastic games. In *Proceedings of the National Academy of Sciences* (1953), vol. 39, pp. 1095–1100.

[88] Shoham, Yoav, Powers, Rob, and Grenager, Trond. If multi-agent learning is the answer, what is the question? *ARTIFICIAL INTELLIGENCE 171* (2007).

[89] Simon, H. A. Nearly-decomposable systems. In *The Sciences of the Artificial* (1969), pp. 99–103.

[90] Simon, H. A. *The Sciences of the Artificial*. MIT Press, 1981.

[91] Sims, Mark, Goldman, Claudia, and Lesser, Victor. Self-Organization through Bottom-up Coalition Formation. In *AAMAS'03* (2003), pp. 867–874.

[92] Singh, Satinder, Kearns, Michael, and Mansour, Yishay. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (2000), Morgan, pp. 541–548.

[93] Singh, Satinder P., Jaakkola, Tommi, and Jordan, Michael I. Learning without state-estimation in partially observable markovian decision processes. In *In Proceedings of the Eleventh International Conference on Machine Learning* (1994), Morgan Kaufmann, pp. 284–292.

[94] Singh, Satinder P., Jaakkola, Tommi, Littman, Michael L., and Szepesvari, Csaba. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning 38*, 3 (2000), 287–308.

[95] Soni, Vishal, and Singh, Satinder P. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI* (2006).

[96] Stone, Peter, and Veloso, Manuela. Team-partitioned, opaque-transition reinforcement learning. In *Autonomous Agents'99* (1999), pp. 206–212.

[97] Stranders, Ruben, Farinelli, Alessandro, Rogers, Alex, and Jennings, Nicholas R. Decentralised coordination of mobile sensors using the max-sum algorithm. In *IJCAI* (2009), pp. 299–304.

[98] Sutton, Richard S. Learning to predict by the methods of temporal differences. *Mach. Learn. 3*, 1 (1988), 9–44.

[99] Sutton, Richard S. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference (1990) on Machine learning* (San Francisco, CA, USA, 1990), Morgan Kaufmann Publishers Inc., pp. 216–224.

[100] Sutton, Richard S., and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[101] Tan, Ming. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning* (1993), Morgan Kaufmann, pp. 330–337.

[102] Tangamchit, P., Dolan, J., and Khosla, P. Learning-based task allocation in decentralized multirobot systems. In *DARS'00* (2000), pp. 381–390.

[103] Tao, Nigel, Baxter, Jonathan, and Weaver, Lex. A multi-agent policy-gradient approach to network routing. In *ICML '01* (2001), pp. 553–560.

[104] Tasaki, M., Yabu, Y., Iwanari, Y., Yokoo, M., Tambe, M., Marecki, J., and Varakantham, P. Introducing communication in dis-pomdps with locality of interaction. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (2008), vol. 2, pp. 169 –175.

[105] Taylor, Matthew E., and Stone, Peter. Cross-domain transfer for reinforcement learning. In *ICML* (2007), pp. 879–886.

[106] Taylor, Matthew E. and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research 10*, 1 (2009), 1633–1685.

[107] Team, NWB. Network workbench tool. In *http://nwb.slis.indiana.edu* (2006).

[108] Tesauro, Gerald. Practical issues in temporal difference learning. *Machine Learning 8* (May 1992), 257–277.

[109] Tesauro, Gerald. Online resource allocation using decompositional reinforcement learning. In *AAAI* (2005), pp. 886–891.

[110] Tsitsiklis, John N., and Sutton, Richard. Asynchronous stochastic approximation and q-learning. In *Machine Learning* (1994), pp. 185–202.

[111] Tuyls, Karl, Hoen, Pieter Jan, and Vanschoenwinkel, Bram. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Autonomous Agents and Multi-Agent Systems 12* (January 2006), 115–153.

[112] Tuyls, Karl, Verbeeck, Katja, and Lenaerts, Tom. A selection-mutation model for q-learning in multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems* (New York, NY, USA, 2003), AAMAS '03, ACM, pp. 693–700.

[113] Urgaonkar, Bhuvan, and Shenoy, Prashant. Sharc: Managing cpu and network bandwidth in shared clusters. *IEEE Trans. on Parallel and Distributed Systems (TPDS) 14*, 11 (2003).

[114] Varakantham, Pradeep, Tambe, Milind, and Yokoo, Makoto. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI* (2005), pp. 133–139.

[115] von Neumann, John, and Morgenstern, Oskar. *Theory of Games and Economic Behavior*. John Wiley and Sons, 1944.

[116] Wang, Xiaofeng, and Sandholm, Tuomas. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Advances in Neural Information Processing Systems* (2002), MIT Press, pp. 1571–1578.

[117] Watkins, C. J. C. H., and Dayan, P. Q-learning. *Machine Learning 8*, 3/4 (1992), 279–292.

[118] Weinberg, Michael, and Rosenschein, Jeffrey S. Best-response multiagent learning in non-stationary environments. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 506–513.

[119] Wilson, Aaron, Fern, Alan, Ray, Soumya, and Tadepalli, Prasad. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML* (2007), pp. 1015–1022.

[120] Wolpert, David, and Tumer, Kagan. Optimal payoff functions for members of collectives. In *Advances in Complex Systems* (2001).

[121] Zafar, Huzaifa, Lesser, Victor, Corkill, Daniel, and Ganesan, Deepak. Using Organization Knowledge to Improve Routing Performance in Wireless Multi-Agent Networks. In *Proceedings of Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)* (Estoril, Portugal, 2008), Parkes Padgham and Parsons Mller, Eds., IFMAAS, pp. 821–828.

[122] Zhang, Chongjie, Abdallah, Sherief, and Lesser, Victor. MASPA: Multi-agent automated supervisory policy adaptation. In *University of Massachusetts Amherst Computer Science Technical Report #08-03* (2008).

[123] Zhang, Chongjie, Abdallah, Sherief, and Lesser, Victor. Integrating organizational control into multi-agent learning. In *AAMAS'09* (2009).

[124] Zhang, Chongjie, and Lesser, Victor. Multi-agent learning with policy prediction. In *AAAI'10* (2010).

[125] Zhang, Chongjie, and Lesser, Victor. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *AAAI'11* (2011).

[126] Zhang, Chongjie, Lesser, Victor, and Abdallah, Sherief. Self-organization for coordinating decentralized reinforcement learning. In *University of Massachusetts Amherst Computer Science Technical Report UM-CS-2009-007* (2009).

[127] Zhang, Chongjie, Lesser, Victor, and Abdallah, Sherief. Self-organization for dynamically supervising distributed learning. In *University of Massachusetts Amherst Computer Science Technical Report UM-CS-2009-007* (2009).

[128] Zhang, Chongjie, Lesser, Victor, and Abdallah, Sherief. Self-organization for coordinating decentralized reinforcement learning. In *AAMAS'10* (2010).

[129] Zhang, Chongjie, Lesser, Victor, and Shenoy, Prashant. A Multi-Agent Learning Approach to Online Distributed Resource Allocation. In *IJCAI'09* (2009).

[130] Zhang, Haizheng, and Lesser, Victor. A reinforcement learning based distributed search algorithm for hierarchical content sharing systems. In *AAMAS'07* (2007).

[131] Zhang, Ying, Liu, Juan, and Zhao, Feng. Information-directed routing in sensor networks using real-time reinforcement learning. *Combinatorial Optimization in Communication Networks 18* (2006), 259–288.

[132] Zhao, Jieyu, and Schmidhuber, Jurgen. Incremental self-improvement for life-time multi-agent reinforcement learning. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (1996), MIT Press, pp. 516–525.

[133] Zinkevich, Martin. Online convex programming and generalized infinitesimal gradient ascent. In *ICML'03* (2003), pp. 928–936.