Coordination in Multi-Agent Systems

by

Chen Gang

A THESIS SUBMITTED TO
NANYANG TECHNOLOGICAL UNIVERSITY
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

April, 2005

Acknowledgement

I would like to thank many people for their support, encouragement, and great help during my years as a PhD student at Nanyang Technological University.

First and foremost, this thesis represents a great deal of time and effort not only on my part, but on the part of my supervisor Dr. Yang Zhonghua and my co-supervisor Dr. He Hao. My earnest thanks go to Dr. Yang for his guidance, mentoring, and support. He absolutely deserves a great deal of credit for being a sagacious and patient critic of all my ideas. He helped me shape my research from day one and encouraged me to achieve the best of my ability. Without Dr. Yang, this thesis would not have happened. My special thanks go to Dr. He, who spent so much time revising my papers and examining my new ideas. His suggestive advices pushed me through inevitable setbacks and helped me fulfill my potential.

I'd also like to thank the Singapore Institute of Manufacturing Technology (SIMTech) for the full financial support of my PhD research. I want to give my special thanks to Mr. Goh Kiah Mok as he is always friendly and willing to help for any of my difficulties.

Let me not forget those people whose job was not to teach me, but from whom I've learned a great deal. Thanks my friends and labmates, who have kept my spirits up and my life interesting, and wonderful technical staffs, who kept things working behind the scenes.

My family has always been a great blessing and source of joy to me over the years. My parents and my sister have been, and will continue to be, there for me at all times. I offer my deepest thanks to them for their emotional support, which has pushed me to carry on.

Table of Contents

List of Figures

List of Tables

Summary

Coordination, as an art of managing interdependencies among activities, is one of the central research issues in multi-agent systems. This thesis addresses agent coordination in task-oriented domains where multiple agents cooperating with each other in order to achieve a series of tasks. The research is performed with the belief that the multi-agent system can be designed to exhibit desirable properties or functions and each agent makes its local coordination decisions based upon its knowledge of the environment and other agents.

To model the information essential to agent coordination, this thesis proposes a Fuzzy Subjective Task Structure (FSTS) model. Through this model, the agent coordination problem is viewed as a Decision-Theoretic Planning (DTP) problem, to which reinforcement learning can be applied. Two learning algorithms, "coarse-grained" and "fine-grained" are presented to address agent coordination behavior at two different levels. The "coarse-grained" algorithm operates at one level and tackles hard system constraints, while the "fine-grained" at another level and for soft constraints. These learning algorithms are formally proved to converge, and due to the explicit modeling and exploration of the coordination specific information, they are also effective in improving the global system performance.

The two reinforcement learning algorithms assume that each agent has a global system view. An interacting protocol is designed so that each agent, following this protocol, maintains a consistent system-wide state information for effective learning. The properties of the protocol are derived and theoretically analyzed.

Without focusing only on reinforcement learning techniques, this research also proposes a bio-inspired approach to multi-agent coordination. The identification of coordination decisions is viewed as an adaptive process. A dynamic coordination model inspired by biological metabolic system is proposed. The model establishes an analogy between task-oriented domains and a simplified metabolic system. Agent

coordination is achieved as every agent performs iteratively a dynamic optimization process, which utilizes explicitly the global dynamics represented through the metabolic system model.

All research results and contributions presented in this thesis are experimentally evaluated to be effective. The practicality of the research results is investigated in the context of a pathology lab system and a shop floor manufacturing system.

Chapter 1

Introduction

1.1 Motivation

In the past decade, an increasing number of computer systems are being built under the multi-agent system paradigm. While there lacks generally accepted definition of "agent", in this thesis, an agent is an entity with perceptions, goals, communication capabilities, actions, and domain knowledge, situated in an environment [149]. The ways it maps from perceptions over time to actions are called its "behaviors". Agent technologies provide an overarching framework for bringing together miscellaneous AI disciplines that are necessary for designing and building intelligent systems [201]. Since any individual agent may not be eligible to solve real-life problems by itself, the use of multiple agents is essential for the successful employment of agent technologies. Multi-agent systems, instead of single agent systems, have been advocated as the next generation model of engineering complex, distributed systems [149].

Multi-agent systems are important to research for a number of reasons. On the one hand, they help scientists propose and test their theories about existing systems. For example, on a macroscopic level, agents may represent social entities for the purpose of investigating social and economic systems. Taking a rather microscopic view, agents can model biological structures (e.g. cells) that come together to form a living organism. On the other hand, multi-agent systems possess unique characteristics that facilitate the more effective synthesis of engineering systems in such fields as distributed computing, decentralized control, and computer supported collaboration systems. Multi-agent systems are often required due to spatial or geographic distributions, or in situations where centralized control is impractical. Even when a distributed solution is not compulsory, multiple agents may still serve as a scalable and flexible approach towards complex real-life problems.

The use of multiple agents raises certain research issues not considered by single-agent systems. Agents should be able to deal with the uncertainties they

have about themselves, the relations with other agents, and the environment. More importantly, as agents depend upon each other, either through the interactions of their activities or the relations of their goals, they have to coordinate their behaviors and manage their interdependencies. As Durfee pointed out, in order to solve practical problems, agents must work in a coherent manner, exchange information with each other, manage the use of shared resources, avoid conflicts, predict and resolve arising conflicts, etc. [107]. It has been widely evidenced that *the missions of multi-agent systems can be achieved only by recognizing that agents are living in an interdependent environment with conflicts, overlapping goals, and mutual dependencies.*

Agents need to coordinate their behaviors for at least three main reasons [148]: (1) there are dependencies between agents' actions; (2) there are needs to meet global constraints; and (3) the entire problem cannot be solved by any agent alone. Whenever agents meet those conditions, they need to employ certain coordination mechanisms in order to ensure the coherent running of the overall system. As has been identified by Nwana *et. al.*, coordination mechanisms seek to solve five main issues [203]: (1) promote network coherence (optimize the performance of a group of agents); (2) facilitate task and resource allocation among agents; (3) recognize and resolve conflicts of agents' local goals, facts, beliefs, or behaviors; (4) determine the organizational structure of multiple agents; and (5) make sure that all the necessary portions of the overall problem are included in the activities of at least one agent. Due to the interdisciplinary nature of the coordination research [190], all sorts of AI and computing techniques, including logic, multi-agent planning, case-based reasoning, distributed constraint management, game theory, and machine learning, have been explored to construct effective coordination mechanisms.

Traditionally in Distributed Artificial Intelligence (DAI), multi-agent systems are often characterized by the sophistication of their constituent agents [83]. The research focuses on designing systems that employ highly complex local decision makers [177]. Each decision maker is assumed to be an intelligent problem solver. The coordination mechanism utilized by these problem solvers is usually based on a large amount of high-level knowledge, including information about the desires, intentions, and plans of other agents, as well as the goals of the entire system. But as supported by many researchers, decision making based on local information seems more efficient than centralized reasoning using the complex knowledge about other agents [86]. Specifically, one of the on-going efforts has been to identify conditions that lead agents, who reason locally, to choose to act in particular ways such that the society of agents exhibit certain desirable properties.

This thesis considers groups of agents working in a *task-oriented* environment

where agents achieve their goals by completing tasks [230]. The proposed coordination techniques assume that the multi-agent systems possess three main characteristics:

- **Cooperative**: agents are inherently cooperative and share *identical* objectives. They are assumed to be honest and only exchange information which they believe to be true.
- **Free communication**: communication is performed in a peer-to-peer manner. Any agent can communicate with the rest of agents in the system through sending messages.
- **Limited capability**: individual agent may not be capable of finishing any tasks alone. They need to cooperate so that each agent contributes to the completion of a task by performing several actions. The inter-relation among agents is determined by the inter-related actions they perform.

Though it is possible for agents to have *conflicting* objectives, such situations would lead to much more complicated design of coordination mechanisms. For many applications, it is an appealing decision to build agents that remain cooperative to each other. Meanwhile, due to the booming development of hardware technologies, free communication also serves as a practical assumption in many real-life situations. Previous researches have addressed domains with similar characteristics. For example, Durfee proposed the Partial Global Planning (PGP) algorithm for agent coordination in the distributed vehicle monitoring testbed (DVMT), where the three characteristics above have been explicitly utilized [100].

The research reported in this thesis is performed with the general belief that the multi-agent system can be designed to exhibit desirable properties or functions and each agent makes its local coordination decisions based upon its knowledge of the system. The primary research goal is to investigate the effectivenss coordination mechanisms or techniques in improving the global system performance while each agent makes its local decisions. The key application is to employ agent technologies in distributed control environments. All proposed coordination techniques are used to enhance the cooperation among a network of distributed controllers. Accordingly, the improvements in the coordination techniques are evaluated in terms of the improvement in the agents' task completing performance. This is reflected by the prototyping systems described throughout the thesis. To further evaluate the effectiveness of the suggested coordination approaches, general-purpose optimization and searching algorithms (e.g the genetic

algorithms) are used as benchmark methods to compare the advances made in this thesis, as evidenced in Sections 4.7 and 6.5.

Generally, this thesis focuses mainly on the theoretical aspects of agent coordination. Two reinforcement learning algorithms and a dynamic optimization algorithm are proposed to coordinate multiple agents and to constantly improve the global system performance. The thesis provides theorems and propositions to justify the effectiveness of the algorithms, which can be further used as general guidance for practical applications of these techniques.

1.2 Objectives

The main research objective of this thesis is to investigate and develop coordination techniques for multi-agent systems. In particular, this thesis seeks to find effective coordination approaches that enable agents to accomplish their missions in a changing and interdependent environment. There are a number of ways to contribute to this objective. In the context of objective coordination, research emphasizes multi-agent system architectures, coordination languages and run-time systems [247]. The central issue is to provide methodologies and techniques that help system designers construct their multi-agent systems where coordination is treated as first-class citizen [213]. On the other hand, under the context of subjective coordination, research is performed with respect to agents' logical behaviors and intra-agent dependencies (i.e. dependencies that result from the particular mental states of each agent). This thesis focuses on subjective coordination. Coordination techniques are proposed to improve agents' decision-making where

- Agents operate in a distributed environment and real-time decision-making is desirable.
- Agents are inherently distributed. No centralized control in the form of designated coordinator agents will be adopted.
- The system performance, which is expected to be improved through agent coordination, is defined globally over the collective behaviors of all participating agents.

In order to effectively improve the system performance, at least three design issues should be handled properly:

- How to model the problems to be solved by multiple agents, and how to represent agents' subjective beliefs and intentions with respect to these problems?

- How can agents make coordinated decisions without taking a painful reasoning process?
- What kind of agent interactions may occur and how can they be handled by coordination techniques?

The three design issues have all been covered in this thesis. To tackle the first one, a model that describes general coordination problems in task-oriented domains has been proposed. Without using complex reasoning techniques such as distributed planning or constraint-based reasoning, task-oriented domains are viewed as dynamical systems and agent coordination is considered as a *dynamic optimization* process. There are two main tools of dynamic optimization: *dynamic programming* and *calculus of variations* [42]. Following the former approach, agent coordination is treated as a Markov Decision Process. The reinforcement learning method is presented in the thesis to help agents obtain good global performance.

Calculus of variations is explored later in the thesis to coordinate agents in environments modeled as metabolic systems. The two approaches achieve agent coordination with two different perspectives. Together, they give a more complete picture of agents' coordinated decision-making in dynamical systems [94, 140]. The third design issue is covered in the thesis through a protocol used to manage interactions among learning agents for effectiveness learning.

1.3 Contributions

This thesis makes four distinct contributions to the field of agent coordination in multi-agent systems.

- **Fuzzy Subjective Task Structure Model (FSTS)**. The FSTS model is proposed to describe agent's subjective beliefs and preferences in an uncertain environment. With this model, the information essential to agent coordination is effectively and explicitly modeled and incorporated by each agent to guide their local decision-making processes. Since agents are often uncertain about themselves and their environment, a model to take those uncertainties into consideration is necessary. Traditionally, in task-oriented domains, agents have been assumed to have definite knowledge about what they should and what will be the outcome. But in real-life applications, recognizing the uncertainties in their knowledge could be very important for agents to coordinate their behaviors. For this reason, FSTS model employs fuzzy logic techniques to model those uncertainties that are essential to agent coordination. The research on FSTS has been published in [58, 60].

- **Agent Coordination via Reinforcement Learning**. Using the FSTS model, the agent coordination problem in task-oriented domains is viewed as a Decision-Theoretic Planning (DTP) problem, to which reinforcement learning algorithms can be applied. Two learning algorithms, "coarse-grained" and "find-grained", are presented to address agent coordination process at two different levels. The "coarse-grained" algorithm operates at one level and tackles hard system constraints, while the "fine-grained" at another level and for soft constraints. These learning algorithms are formally proved to converge, and due to the explicit modeling and exploration of coordination specific information, they are also effective in improving the global system performance. The research on learning based coordination approaches has been published in [59, 62].

- **Protocols for Maintaining System State Information in a Multi-agent Environment for Effective Reinforcement Learning**. One fundamental issue in agent coordination via reinforcement learning is how to deal with the limited local knowledge of an agent in order to achieve effective learning. In this thesis, effective learning is achieved by requiring agents to follow an interacting protocol so that a consistent system-wide state information is maintained. The properties of the protocol are derived and theoretically analyzed. A distributed protocol that satisfies these properties is also presented. This research has been published in [63].

- **Agent Coordination as Dynamic Metabolic Systems**. Instead of using reinforcement learning, a bio-inspired approach to multi-agent coordination is proposed. In this approach, the identification of proper coordination decisions is viewed as an adaptive process. To facilitate this view, a dynamic coordination model, which is inspired by biological metabolic systems, is presented. The model establishes an analogy between task-oriented domains and a simplified metabolic system. Agent coordination is achieved as every agent performs iteratively a dynamic optimization process, which utilizes explicitly the global dynamics represented through the metabolic system model. This research work is reported in a paper that is currently under review by the International Journal of Software Engineering and Knowledge Engineering [61].

All research results and contributions presented in this thesis are experimentally evaluated to be effective. The practicality of the research results is investigated in the context of a pathology lab system and a shop floor manufacturing system.

1.4 Organization of the Thesis

A brief description of each chapter is as follows.

- **Chapter 2** summarizes the major research activities in the field of agent coordination technologies. It highlights the necessity for coordination in multi-agent systems and reviews various coordination techniques in terms of their design perspective (i.e. objective or subjective coordination) and flexibility.

- **Chapter 3** describes the Fuzzy Subjective Task Structure (FSTS) model, which provides a conceptual foundation for modeling and coordinating agents' behaviors in task-oriented domains.

- **Chapter 4** presents both the "coarse-grained" and the "fine-grained" reinforcement learning algorithms for agent coordination. The theoretical analysis of the algorithms and the experimental evaluation of their effectiveness in a pathology lab system are also included in this chapter.

- **Chapter 5** investigates the desirable properties of an agent interaction protocol, which will help agents maintain a consistent system-wide state information for effective reinforcement learning. A distributed protocol that satisfies these properties is presented. The experimental evaluations are conducted for a well-known test-case (i.e. *pursuit game*) in the context of two reinforcement learning algorithms.

- **Chapter 6** proposes a new coordination approach inspired by biological metabolic systems. A dynamic coordination model is described. The dynamic optimization process performed by each agent is further detailed. Numerical analysis has been performed and demonstrates the effectiveness of this approach. The practicality of the approach is examined in a shop floor manufacturing system.

- **Chapter 7** summarizes the contributions of this thesis and outlines the most promising directions for future work.

Chapter 2

Coordination Technologies in Multi-Agent Systems

2.1 Introduction

As one of the central research issues of multi-agent systems, agent coordination has attracted growing research interests in the last two decades. Numerous theories, methodologies, techniques, algorithms, and industrial standards have been proposed in the effort of constructing fully-coordinated multi-agent systems. Coordination is considered as a fundamental capability agents have to decide their own behaviors in the context of the activities of other agents [102]. Nevertheless, the notion of coordination is often vaguely understood and is of interdisciplinary nature [190].

From a broad view, coordination is defined as "*the act of working together harmoniously*" [190]. It has been studied by researchers in diverse disciplines, such as sociology, biology, operations research, organizational theory, social psychology, and computer science. For example, sociologists have investigated the interdependencies of social behaviors and the self-maintenance of social systems with regard to these relationships [215]. Economics have studied coordination in various markets populated by profit-maximizing firms [189, 289]. Organization theorists have examined the mechanisms by which human beings coordinate with each other to form groups of all sizes [271].

Interestingly, these researches lead to one intuition, that is, good coordination is nearly *invisible* and often its importance only becomes clear when it is lacking [190]. This phenomenon has been explicitly studied by Laszlo as he pointed out that agents (e.g. people) live through their practices and tacit knowledge so that the powerful things (e.g. coordination) are those that are effectively *invisible* in use [173]. For example, in biology, the existence of an organism, such as that of our human beings, depends on the precise coordination of individual cells, even though these cells act in a seemingly purposeless manner. Essentially, as inspired

by the works of Malone, Gelernter, and Jennings, coordination is believed to be the art of *managing the interaction and interdependencies between entities of a system*, be the entities agents, processes, cells, or whatever [118, 148, 190].

This notion of coordination emphasizes two aspects, the interdependencies between entities and the processes managing these interdependencies. In the realm of multi-agent systems, the interaction between agents covers both subjective interdependencies (e.g. commitment and social conventions [148]) and objective interactions (e.g. communication act) [247]. Researchers in the areas such as Distributed Artificial Intelligence (DAI) prefer an *agent-oriented* approach [35]. The interactions are described by agents' subjective mental states, such as their beliefs, desires, and intentions [287], while the actual means and the place of the interaction are assumed available *a priori*. On the contrary, researchers in the areas of distributed computing and software engineering focus more on identifying and modeling the interaction settings [234].

Generally, any interdependencies between agents can be approached with either of two different viewpoints: dependencies rely on the objective composition of multiple agents, and dependencies resulting from the particular mental states of each agent. Inspired by these two viewpoints, the coordination technologies proposed in the literature can be roughly classified into two categories. They are

- Objective Coordination Technologies: the technologies to manage inter-agent dependencies. The central issue is around the configuration of multi-agent systems through the basic interaction techniques. Methods are provided to organize the interacting place in order to facilitate the construction of complex coordination processes and meet certain performance requirements.

- Subjective Coordination Technologies: the technologies to manage intra-agent (subjective) dependencies towards other agents. The central issue is to develop coordination processes that access and alter agents' subjective mental states and help them make coordinated local decisions. Coordination processes can be further classified as designed from either *bottom-up* or *top-down*. The details will be covered in the subsequent sections of this chapter.

In general, subjective coordination relies upon objective coordination technologies as the former presumes the existence of the latter. Coordination processes satisfying subjective coordination requirements must indeed access the mechanisms for objective coordination. As Rowstron indicated [234], when designing a multi-agent system, the first step is to identify which objective dependencies are presented and how they can be handled. It is only after this step that all

subjective dependencies can be identified and addressed. The objective and subjective coordination technologies are interrelated and equally important [205]. The distinction between the two approaches is essentially a *methodological* one [207]. The key issue is how they could be used together effectively to tackle complex multi-agent interactions.

Coordination technologies may also be characterized from another dimension concerning their degree of flexibility. Flexibility refers to the ability of agents to reorganize their behaviors in order to fit the information fed back from the environment and other agents [173].The more flexible, the more robust the coordination technologies will be to achieve intended operations under changing conditions. Specifically, the following degrees of flexibility are distinguished:

- Fixed interaction means and coordination processes. Agents use basic interaction operations with pre-defined semantics [46] or follow coordination processes with pre-determined sequence and outcomes.
- The semantics of the interaction operations are altered according to the temporary feedback from the environment and the multi-agent system. The coordination processes provide subroutines with the choice determined by the feedback as well.
- The interaction patterns among agents *emerge* from the agents' local decisions. There is no finely designed coordination processes or subroutines. Only agents' local decision-making policies remain fixed.
- Optimizing dynamic programming with the capability to reprogramme agents' decision-making policies in regard to the need signaled in the information feedback. Agents are able to modify their interaction patterns to better adapt to the changing environment.

The above four degrees take us from fully rigid coordination technologies to fully flexible technologies. Although flexibility brings agents' adaptability, the reliability and practicability (e.g. computational complexity for coordination) may suffer. The more flexible, the more difficult these technologies can be implemented and depended on for reliable functions. There is no single coordination solution that fits all situations [104, 106].

Overviews and taxonomies on agent coordination technologies are numerous [89, 105, 178, 203, 213, 257, 262]. Each organizes the whole picture with its own way. In this chapter, the focus is mainly on the above two dimensions. The discussion is limited to software multi-agent systems. The breakdown of this chapter is as follows. Section 2.2 introduces objective coordination technologies. Section

2.3 further describes subjective coordination technologies, which have been broken into four parts: agent communication, multi-agent platform, top-down coordination technologies, and bottom-up coordination technologies. Section 2.4 finally summarizes this chapter. Subsequent chapters include discussions of related work that most closely pertains to their particular topics. This chapter does not repeat those discussions.

2.2 Objective Coordination Technologies

Objective coordination focuses on the management of inter-agent dependencies. Coordination is considered as enabling technologies that allow software agents to join in a concurrent ensemble [232]. It is basically achieved through *normative activities* by certain part of a multi-agent system on behalf of the system designers, such as a coordination medium provided by an infrastructure [207]. Generally, the normative activity refers to the standard operations, conventions or assumptions accepted by all agents in the system. It supports the organization of a multi-agent encounter by describing how the environment is organized and by handling the inter-agent interactions [247].

Environment refers to a place where multiple agents integrate their activities. Two types of environment organizations are distinguished: *implicit* and *explicit* organizations [247]. An *implicit* organization does not model the environment as it is imposed by the logical structure on which a multi-agent system evolves. As an example, the network structure may be determined indirectly through the nodes of the network by network-aware agents [45]. For *explicit* organization, an environment model is provided. But it does not necessarily reflect the underlying logical structure where the agent lives. For instance, in a multi-agent manufacturing system, although the environment is inherently continuous, agents will convince themselves that they live in a discrete event environment [184].

Interactions exist between an agent and its surroundings. The agent relates its behaviors with the environment through perception and specific actions. There's no direct communication between the two bodies. In contrast, interactions between agents are usually based on specific communication means, following either of the four basic paradigms [66]:

1. **Peer-to-peer Communication**: Messages are sent directly from one agent to another agent.

2. **Broadcast**: Messages are sent from one agent to every agents in a multi-agent system.

3. **Group Communication**: A message is sent to a specific group of agents.

4. **Generative Communication**: Communication is achieved through utilizing a coordination medium, such as a tuple space [233]. Agents put tuples into the space, which are read later by other agents. The communication is fully uncoupled. The messages are read independent of the time when they are generated [115].

Besides direct communication, agents can also interact with each other through the mediation of the environment. This is realized in two ways: (1) by changing the environment perceptible by other agents (e.g. *stigmergy* [127]); and (2) by changing the expected effects of other agents' actions [139]. For example, in the *follow-me* application scenario, a guider agent may leave traces in the environment, which will be utilized later by some follower agents [206]. In order to describe, support, and manage all those kinds of interactions, the main concern of objective coordination is around the problem of developing coordination models, languages, and run-time systems from a purely computational perspective. They are discussed in the following subsections.

2.2.1 Coordination Models

From a programming languages perspective, a complete programming model can be built out of two separate pieces – the computation model and the coordination model. The computation model allows programmers to construct a single thread, step-at-a-time computational activity. The coordination model is the *glue* that binds separate activities into an ensemble [119]. This statement is better exemplified with the equation below

$$Programming = Computation + Coordination$$

In the context of software engineering, this formulation is extended such that [200]

$$System = Components + Coordination$$

With this view, coordination model serves as a formal framework to express the interactions among software components in a multi-component system [71]. It has to address three important issues: (1) how do agents synchronize their work; (2) how do agents communicate; and (3) how their activities are started [232]. These issues are dealt with in a variety of ways. Generally, coordination models fall into either of two major categories: *data driven* or *control driven* [212].

In a data driven coordination model, coordination is achieved by concurrent access to one or more content-addressable tuple spaces. Agents communicate with each other only indirectly through the shared data. Since the shared tuple spaces

are implemented independently from each agent, data driven models achieve decoupling of agents both in space and in time. In control driven models, agents communicate with each other in a point-to-point manner by means of well-defined interfaces (i.e. input and output ports). The system evolves dynamically by generating and receiving control events. To separate computation and coordination, control-driven models provide coordination languages that treat agents as black boxes linked by their interfaces.

One notable data-driven coordination model is *Linda* [5,118]. It introduces the notion of *uncoupled communication* whereby agents exchange data through shared *tuple spaces*. A tuple space contains tuples, which enable structured representation of application data. Linda provides primitives as basic means for agents to access tuple spaces. A tuple can be stored in a tuple space by an agent performing the *out* primitive. Two primitives are provided to retrieve data from the tuple space: *in* and *rd*. Data retrieval is facilitated by a pattern matching mechanism in order to identify data of desired properties. Executing the primitive *in* will remove the matched tuple, while primitive *rd* leaves the tuple untouched in the tuple space. Both *in* and *rd* are *blocking*. They do not return as long as there is no matching tuple in the tuple space. Finally, the primitive *eval* will add additional agents to the multiset of active agents [46]. It has been demonstrated that Linda is able to express all major styles of coordination in concurrent computing environment [54]. One drawback of Linda is the lack of formal semantics. An attempt to compensate this shortage can be found in [46].

Many enhancements of the original Linda model have been proposed recently, following the three directions: (1) extending the set of coordination primitives; (2) adding programmability to the semantics of the primitives; and (3) modifying the original model structures [233]. In many cases, the primitive set is enlarged by introducing two additional primitives: *inp* and *rdp* [232]. Unlike primitives *in* and *rd*, *inp* and *rdp* do not block the execution of the calling agent. If no matched tuples exist in the tuple space, *inp* and *rdp* return instantly with a false indication. Programmability is provided for certain variations of the Linda model, such as LuCe and TuCSoN [93, 208], to further uncouple the coordination and computation aspects. For example, in LuCe, *specification tuples* are used to describe tuple spaces' behaviors [93]. This ability enables the tuple spaces to be reprogrammed so as to bridge between the different representation of information shared by agents, or to embed new laws for agent coordination [233]. Finally, to adapt to particular application environments, several coordination models enhance Linda by modifying its model structure, such as Laura and Lime [219,273]. For instance, Laura extends tuple spaces into *service spaces*, which are a collection of *forms* [273]. Forms contain description of service-offers, service-requests,

or service-results. Laura is designed to support the identification and utilization of services provided by agents. It has been used to coordinate applications based on PageSpace, which is a reference architecture for distributed applications [70].

Another data driven coordination model substantially different from Linda is the Gamma (General Abstract Model for Multiset mAnipulation) coordination model [15]. It is based on the concept of *multisets*. Agents in Gamma takes the chemical reaction formalism. The computation involves substituting those elements in the multisets satisfying certain *reaction conditions* with the products of reactions. Besides this basic procedure, extensions of Gamma have been proposed to improve the original model with structured multisets, parallel and sequential operators, and high order functionalities [232]. Compared with Linda, Gamma fails to win the majority favor of software researchers, especially when agents live in a dynamic open environment such as the Internet.

In addition to data-driven models, control-driven coordination models have been used to compose multiple agents as well. One example is the Ideal Worker Ideal Manager (IWIM) model, which is built upon the Manifold coordination language [8, 9]. IWIM distinguishes two different types of agents: *managers* (or *coordinators*) and *workers*. A manager is responsible for controlling the communications in a group of worker agents. A worker agent, on the other hand, is completely unaware of whom it exchanges information with. From a coordination perspective, every worker agent is a *black box* with well-defined connection *ports*, through which it exchanges information with the rest of the world [67]. Each port is used for information transfer in only one direction: either input port or output port. The interconnection of ports of different agents forms streams. A stream connects an information producer agent with an information consumer agent to enable actual information flow. Activity in IWIM is *event driven*. A manager waits for the occurrence of some events, which trigger it to enter a particular *state* and set up or break off certain streams. It will remain that state until further events come. Contrary to data-driven coordination model, in IWIM, communication is performed in a point-to-point manner. IWIM facilitates a clear separation of coordination and computation as the managers, who take charge of the interactions among worker agents, are programmed separately using Manifold [9].

Comments:

The main research question in the field of coordination model research concentrate on how to establish communication among agents. In data-driven

models, agents communicate by using primitives to post and retrieve data from shared tuple spaces. This effectively implies that the coordination code of an agent is intermixed with its computation code (domain activity) [67,232]. It is up to system designers to seek a clean separation between coordination and computation. As a consequence, the reuse of the code for both computation and coordination may not be easy.

On the other hand, by performing communication in a point-to-point manner and utilizing designated coordinator agents, control-driven coordination models facilitate a more clear separation of coordination and computation. The drawback lies in the requirement of new programming languages and development tools [9]. Besides, control-driven models do not provide strong support for distributed systems to work in open dynamic environment, such as the Internet. They are uncompetitive in developing systems and network dependent solutions [232].

All coordination models enable the construction of either fixed or adaptive coordination processes through agent communication. They can be used to implement various coordination patterns [85, 132]. For example, the Master/Worker pattern, which is used to manage task assignments [190], can be easily implemented as fixed coordination processes using Linda. As Rossi shows, the implementation contains only a few lines of code [232]. Agents may also exchange their local plans and decisions through tuple spaces, and jointly satisfy certain global requirements [57, 64]. In this case, the coordination process emerges from all agents' activities. It is noted that certain coordination models allow the semantics of the interacting primitives to be re-defined. This ability simplifies the construction of coordination processes with timely responses to information feedback. Nevertheless, the question on how the primitives should be reprogrammed is untouched.

2.2.2 Coordination Run-time Systems

All coordination models require run-time systems to provide the models' functionalities. Among the numerous run-time systems proposed over the last two decades, few have the long-lasting impact as tuple-based run-time systems [118]. In fact, tuple-based systems have been implemented to support parallel computing clusters, workstations in local area networks, and large-scale interactions over the Internet [234]. In this subsection, we focus on these run-time systems.

Tuple-based run-time systems can be subdivided into implementations for parallel computers, for Local Area Networks (LANs), and for Wide Area Networks (WANs) [234]. The major distinction is whether they are *open* or *closed*. A *closed*

run-time system prevents the agents from leaving or joining the system at their own will. Usually, all the source code of agents must be available at the compile time. The compiler performs certain analysis to improve the performance of accessing tuples in the tuple spaces. Most early implementations of Linda are closed implementations [53, 175]. They are used to coordinate processes or agents on parallel computers and LANs. Closed implementations have the ability to conduct agent communications very efficiently by using pre-compiling techniques. However, they are over restrictive and cannot be used in open application environments, such as the Internet.

The first tuple-based run-time system was implemented by Carriero [53]. It is a closed system designed for parallel computers. The compile-time analysis examines the tuples and templates to identify efficient data structures to store tuples. In the shared memory environment, the identified data structures will be placed in the shared memory. While in distributed memory environment, a copy of the data structure is placed within each processing module of the parallel computers. Broadcasting mechanism is applied to synchronize these distributed data structures. The implementation is highly efficient and the same compile-time analysis is adopted by other run-time systems, such as [31]. In [31], Bjornson considers a hash-based mechanism to distribute tuples among multiple processing modules, instead of obtaining several copies of the same tuple. For any given template, a hashing function is used to identify the module in which a matching tuple may reside. The development keeps progressive as other improvements have also been introduced by [55]. In addition to analyzing tuples and templates, the particular use of tuples is recognized and implemented through efficient procedures to further enhance the system performance. Multiple tuple spaces are organized hierarchically and the compile-time analysis is applied to group tuples more efficiently [76].

In accordance to the booming interests in Internet applications, open implementations of tuple-based run-time systems have received increasing research attentions. In open systems, agents need no information about other agents to share tuples. They may leave and join the system as they wish. Because not all agents are available at the compile time, limited analysis of tuples and tuple usage is performable [234]. As a result, the exchange of information between agents is achieved in a less efficient way than those closed run-time systems. Nevertheless, open implementations emphasize on supporting heterogeneity, security, reliability, availability, and other features, which are important in the Internet. The need for performance is not a major driving force in developing the many open run-time systems [99, 111, 199, 273]. Currently, open run-time systems have been used to coordinate agents on LANs and WANs. Within the LAN setting, the distribution of tuples can follow either of the four paradigms: centralized distribution, uniform

distribution, intermediate uniform distribution, and distributed hashing [52].

A new generation of run-time systems have been developed since 1995 to enable Linda in WAN environment [234]. These systems extend the original Linda model to support a variety of tuple accessing primitives with the specific emphasis on the functionalities they provide. Among the many example systems, Jada is the first run-time system developed in Java [69], which integrates the tuple accessing primitives into the Java programming language. JavaSpaces is another run-time system, which works within the Jini infrastructure and provide agents with Linda-based coordination support [115, 182, 283]. The JavaSpaces interface defines the basic tuple accessing operations. The main enhancement introduced with respect to the original Linda are rich typing, matching of subtypes, tuples with behaviors, lease, and transaction management.

Most of the run-time systems designed for WAN are constructed as Internet services. For example, the run-time system of WCL is distributed across multiple servers, each stores an entire copy of the tuple spaces [235]. A control layer is included to monitor how the tuple spaces are being used. It balances the workload across a wide geographical region by instructing the tuple space servers to migrate tuple spaces as necessary. In addition to supporting tuple space operations in a large environment such as the whole Internet, the run-time systems also provide other functionalities to improve fault tolerance [152] and mobility [199]. For instance, in KLAIM, the run-time system is comprised of multiple servers, each represents a geographical location and holds a separate tuple space [199]. KLAIM supports strong migration of agents. An agent can request to migrate at a particular point in its execution. The accessing of tuples is performed in a location-aware manner. Agents have to specify which tuple space they are interested.

Comments:

> The research on run-time systems focuses mainly on the question of how to conveniently provide those coordination functionalities given in the corresponding coordination models. The research shares the same benefits and drawbacks as the coordination models. For the closed implementation of Linda-like systems, there is little use of spatial and temporal separation because all agents are available at compile time. On the contrary, distributed open implementation uses more of the features of Linda to support the interactions between agents which are spatially and temporally separated. Many run-time systems designed for Internet applications have also considered the system stability, fault tolerance, security, and many other features. They can be used to develop highly flexible coordination processes, whereas the mechanisms to achieve coordination through these processes are believed to

be the research topic of subjective coordination.

Despite of the numerous run-time systems constructed, there lack formal approaches to engineer run-time systems in the Internet. Meanwhile, although most of the open run-time systems concentrate on the rich functionalities they provide, the requirement for efficiency may soon become an important issue that affects the future development of tuple-based run-time systems.

2.3 Subjective Coordination Technologies

The intra-agent dependencies are dealt with through subjective coordination technologies. Two approaches to subjective coordination are discernible: the *top-down* approach and the *bottom-up* approach. The *top-down* approach has been widely adopted by researchers in distributed artificial intelligence (DAI) [35]. Coordination in DAI is usually achieved by having each agent perform sophisticated reasoning based on complex knowledge, including the plans and goals of other interacting agents [83]. The coordination strategies are designed *top-down* from a global vantage point. Agents reason about its local actions and the actions of others to try and ensure that the whole community acts in a coherent manner [148].

Coordination technologies in DAI focus on the psychological aspect of agents. They rely more or less on the internal representation of agents' beliefs, desires, and intentions [225]. The community's folklore is that "*more knowledge leads to better coordination*" [237]. Coordination processes supported by these technologies are flexible and are emerged from agents' local decisions. But the decision-making policies remain fixed. It is up to system designers to find effective mechanisms in order to improve the global performance or solve particular problems. As a multi-agent system extends its scale in agent population, heterogeneity, and interdependencies [102], the extraordinary complexity faced by the *top-down* approach is critical.

Being frustrated by the difficulty of designing complex systems from the top down, the *bottom-up* coordination approach has received increasing research attentions recently [32, 75, 96, 217]. Researchers intuitively combine unsophisticated agents together in a *bottom-up* manner to see what emerges when agents work as a group. These agents either follow relatively fixed coordination processes or use simple reasoning mechanisms that may involve only local knowledge accessible to them. This amounts to a sort of iterative procedure: designing a set of agents, observing their group behavior, adjusting the decision policies of each agent or the coordination processes they use, and then returning back to evaluate the impact of the adjustment. In many cases, this procedure of designing multi-agent systems can be automated such that agents alter their local decision-making

policies autonomously according to the environmental feedback (e.g. the global performance achieved). Such automated approach resembles the learning behavior and provides the highest level of flexibility. The coordination processes not only emerge from agents' local decisions, but also adapt to the changing environment as agents deliberately learn their local decisions. Many machine learning methods have been explored for this purpose, such as reinforcement learning [83, 84, 256], neural networks [3], and genetic algorithms [188, 231].

Although amazing results have been achieved by *bottom-up* techniques [84, 187, 191], a high level of human intervention and experimentation still seem inevitable. The high complexity of real-life applications poses another challenge to *bottom-up* coordination approaches, especially when learning methods are used. In most cases, local decision policies are designed out of human intuition. There lack theoretical guarantees of their correct function even in the controlled environments.

Besides *top-down* and *bottom-up* coordination technologies, agent communication from a psychological point of view serves as another topic of subjective coordination research. Communication is considered as actions that affect the mental states of participating agents [81]. In the remaining of this section, agent interaction using speech acts and agent communication languages (ACL) is discussed first. A brief description of multi-agent architectures that support this style of communication follows. The introduction of *top-down* and *bottom-up* coordination technologies is covered afterwards.

2.3.1 Communication Technologies with an Intra-Agent Perspective

Communication has long been recognized as a primitive form of coordination [292]. In the agent community (from a intra-agent perspective), it is treated as *actions* – performed in an attempt to influence other agents appropriately. This is the basic philosophy whereby the *speech act theory* [10] is introduced to describe agents' communication behaviors.

Speech act theory *assumes* that speech actions are performed by agents just like other actions, in the furtherance of their intentions [77, 248]. A number of *performative verbs*, corresponding to the various types of speech acts, have been identified. Examples of such performative verbs are *request*, *inform*, and *propose*. These speech acts are characterized via their preconditions and postconditions, similar to other physical actions [113]. Use the *request* act as an example, the aim of the act is for a *speaker* to get a *hearer* to perform some action. In order to successfully complete the *request* act, the speaker must believe that the hearer is able to perform the requested action. Meanwhile, the speaker must also believe

that the hearer believes it has the ability to perform the action [78]. They serve as the preconditions of the *request* act. When the *request* act has been successfully completed, the postconditions will be that the hearer believes that the speaker believes it wants some action to be performed. Notice that the completion of the *request* act does not imply that the desired action will be performed. The actual execution of any action is based upon agents' *rational* decisions [77]. The *request* act, therefore, is performed by the speaker in an attempt to bring about a state where both the speaker and the hearer intend to perform the desired action.

Speech act theory triggered the development of agent communication languages (ACLs). The DARPA-funded knowledge sharing effort (KSF) has defined two languages for agent communication: the knowledge query and manipulation language (KQML) and the knowledge interchange format (KIF) [168,169]. KQML specifies an envelope format for messages. Each message has a *performative* and a number of parameters, such as the receiver of the message and the ontology used. Various versions of KQML have been proposed during the 1990s [114]. A comprehensive survey of these ACLs can be found in [170].

In addition to the envelope format of messages, the message content is further structured using KIF, which facilitates the exchange of knowledge of particular domains. KIF is closely based on first-order logic. Using KIF, agents can express: (1) properties of things in a domain; (2) relations between things in a domain; and (3) general properties of a domain [292]. KIF includes common components of first-order logic, such as the logic connectives (e.g. *and* and or) and quantifiers (e.g. *forall* and *exist*). It provides a LISP-like notation that allows agents to define new objects, objective function, and the relationships between objects [120]. KIF is designed with the initiative to separate the internal differences between agents from their interactions. It enables agents in an open environment, such as the Internet, to exchange information and conduct flexible coordination processes.

In 1995, the Foundation for Intelligent Physical Agents (FIPA) started to develop standards for multi-agent systems [1]. The focus of this initiative is to develop an ACL (FIPA ACL) [2]. FIPA ACL is analogous to KQML. It defines the envelop format of messages with around 20 performatives (different from KQML). Being noticed that most of the criticisms of KQML are due to the lack of proper semantics, the developers of FIPA ACL managed to give a formal semantics to each performative of FIPA ACL. Their work benefits a lot from Cohen and Levesque's theory of speech acts as rational actions [77]. The semantics are given through a formal representation of *beliefs*, *desires*, and *uncertain beliefs* of agents, as well as the actions agents perform [40]. A *semantic conformance testing* has been recognized and started to be utilized by the FIPA ACL community to ensure that agents respect the *semantics* of the ACL [291].

In addition to ACLs, agent communication and coordination are managed further by other technologies, such as the *Agent-Oriented Programming* and the *COOL* language [16, 250]. Agent-oriented programming can be considered as a specialization of Object-Oriented Programming where agents are viewed as objects with mental states. It contains a series of evolving formalisms. In **Agent-0**, the communication between agents is through several primitives. But in **Agent-K**, the message passing functionality follows the KQML standard [250]. Languages such as COOL use KQML as agent communication language as well [16]. A Finite State Machine (FSM) based model is utilized by COOL to describe agent conversations.

Comments:

> A major research question in this research field deals with the expressiveness and soundness of agent communication languages that formalize the intra-agent perspective of communication. The research assumes that communication is performed in a point-to-point manner among individuals. Although the technologies are designed to support flexible coordination processes, such as negotiation and market-like bidding, the impracticality of engaging complex coordination only by means of direct semantic-driven interaction is widely recognized in the literature [26, 81, 205]. A conceptual framework to combine subjective and objective agent communication therefore seems necessary [207, 226]. For example, the *coordination artifact* concept is proposed to be shared and exploited by a collective of agents for achieving a social objective in [206, 207]. Using this concept, both objective and subjective approaches can be explored under the same coordination context. Agents use their psychological abilities (e.g. reasoning and searching) to identify coordination laws that can be used to manage the interactions among them. These laws are in turn enacted by the objective coordination technologies through the exploitation of the coordination artifacts [207].

2.3.2 Multi-Agent Architectures Supporting Agent Coordination

The initiatives towards building multi-agent architectures are approached at two different levels. At one level, researchers are working towards the standardization of agent interactions, such as the work done by the Knowledge Sharing Effort, OMG, and FIPA [1]. At the other level, the research focuses on realizing the development environment in order to build multi-agent systems, such as RETSINA [263], JADE [22], MASCOT [239], JAFIMA [159], and ZEUS [202]. These development environments provide predefined agent models and tools to

simplify the construction of multi-agent systems. They further allow flexible interactions between agents through the use of ACLs (e.g. KQML and FIPA ACL) and conversation protocols. Additionally, specific agents have been introduced to simplify the management effort. For example, FIPA introduces two types of agents to take the management role: DF Agents and AMS Agents [1]. These agents have been implemented by several FIPA-compliant development environments, such as JADE.

JADE is a multi-agent architecture supporting the development of multi-agent applications in compliance with the FIPA specifications [22]. It offers the following useful features to agent developers:

- FIPA-compliant environment, which includes the Agent Management System (AMS), the Directory Facilitator (DF), and the Agent Communication Channel (ACC).
- Distributed environment. The whole agent platform can be split among several hosts, while only one Java Virtual Machine is active on each host. Agents are implemented as Java threads and are capable of performing multiple tasks concurrently.
- A library of FIPA interaction protocols are ready for use. Interaction protocols are patterns of messages exchanged by two or more agents. The protocols implemented in JADE stretch across a wide range from simple *query-requested* protocol to the well-known *contract net* protocol and the English and Dutch auctions [255].

FIPA does not impose any restrictions on the technologies used for implementing its standard. Possible techniques, such as e-mail based platform, CORBA based platform, and Java multi-threading environment, can all serve as the basis of FIPA-compliant implementations. In addition to FIPA-compliant multi-agent architectures, other systems have been developed to show the particular favor of their designers. One example is RETSINA, which is developed by Carnegie Mellon University [263]. It is first designed for the distributed information retrieval domain [204]. The development of the RETSINA architecture rests on the following considerations:

- Distributed information sources: information sources available on line are inherently distributed.
- Shareability: agents are cooperative in order to share their computational abilities and retrieved information.

- Complexity hiding: information retrieval may involve quite complex coordination of many distributed agents. To improve the usability of the system, interface agents are introduced to take the responsibility of interacting with the end users.

Two types of agents have been identified in RETSINA: task agents and information agents. Task agents facilitate decision-making by forming problem-solving plans. These plans are executed through querying and exchanging information with information agents. To support agent coordination, RETSINA provides distributed scheduling, planning, and communication mechanisms that can be applied for general purpose use. It further simplifies agent construction by identifying the internal structure of agents. Till now, RETSINA has been successfully used to implement several distributed applications including aircraft maintenance, route planning, personal information manager and meeting scheduler, and supply chain management.

Certain multi-agent architectures, such as MASCOT [239], are only designed for specific application domains. Agents' psychological abilities may also be restricted in order to accommodate a huge amount of agents in a single system. For example, the DIET architecture has been proposed to enable the construction of multi-agent systems from a *bottom-up* approach. DIET agents are not assumed to be highly intelligent and can be very lightweight. They do not use complex communication languages such as FIPA ACL. The whole architecture is entirely distributed. All these make it possible to run up to several hundred thousands of agents in a single machine [138].

Comments:

The research on multi-agent architectures aims at providing a set of abstractions that formalize the types of agents, their capabilities, and the interactions among them. In certain architectures, abstractions have been proposed to simplify the task of constructing distributed systems based on common control protocols, such as client-server computing, subsumption, or pipelines of agents [85, 190]. System designers can easily determine at the design time the various responsibilities and interactions of in the system. Abstractions in a coordination process are often described by protocols that agents follow in order to regulate the behavior of the system as a whole. Protocols are characterized by their dynamic nature. Detailed responsibilities and interactions among agents might not be identified until runtime. One typical example is the *contract net protocol*, where the interdependencies among agents vary as the system evolves [255].

Success in multi-agent architectures raises the enthusiasm of adopting the multi-agent paradigm in many application domains. Coordination can be approached more easily by using a multi-agent architecture than being tackled from scratch. Although many architectures support the construction of coordination processes with any degree of flexibility, the issue as how coordinated behaviors can be achieved is seldom touched. It is the main topic of both the *top-down* and the *bottom-up* coordination technologies. Architectures such as DIET are more suitable for bottom-up coordination as they provide a environment that can easily hosts thousands of light-weight agents that interact with each other through simple message passing [138]. On the other hand, multi-agent architectures like RETSINA are more appropriate for top-down coordination as they support complex reasoning and decision-making through planning and scheduling subsystems.

2.3.3 Top-down Coordination Technologies

Based upon the various classifications proposed in the literature [89, 203, 211], top-down coordination technologies can be grouped into four families:

- Coordination through multi-agent planning.
- Coordination through negotiation.
- Coordination through norms and social laws.
- Coordination through mutual modeling and joint intentions.

In the remaining of this subsection, technologies belong to each family will be described one by one.

Coordination through multi-agent planning

In order to avoid inconsistent or conflicting actions, agents build a multi-agent plan that details their future actions and interactions required to achieve their goals. In many respects, multi-agent planning is a specialization of distributed problem solving, where the problem being solved is to design a plan [101]. Regarding to how the multi-agent plan is obtained, this family of coordination technologies are divided into two types: *centralized* multi-agent planning and *distributed* multi-agent planning.

In the centralized multi-agent planning, the plans to be executed by multiple agents are formulated by a centralized planner or coordinator agent. This centralized coordinator agent break the plans into separate threads, possibly with some interactive actions. These separated plan pieces will in turn be passed to agents

that can finally execute them. As plans are formulated in a centralized manner, the potential inconsistencies and conflicts can be identified and eliminated by the coordinator agent before they are ultimately reached by other agents. From a more algorithmical perspective, the major steps of centralized multi-agent planning are summarized as follows [50, 122, 153]:

1. Given a goal description, domain operations, and an initial system state, collect partial or local plans from individual agents.
2. Analyze collected local plans, identify and eliminate potential inconsistencies and conflicts, and form a multi-agent plan (i.e. global plan).
3. Break the multi-agent plan into subplans and insert necessary interaction actions into subplans.
4. Allocate subplans to agents using task-passing mechanisms.
5. Initiate and monitor plan execution. Restart the planning effort when things go unexpectedly.

In distributed multi-agent planning, both the planning process and its results are intended to be distributed. In this case, any agent may never have a multi-agent plan represented in its entirety. Nevertheless, agents' local plans should be compatible and consistent with each other. The literature provide a rich set of methods in order to satisfy this requirement. Two important approaches are *plan merging* and *iterative plan formation* [292]. The merging method proceeds as follows. Given the plans of several agents, the method begins to analyze the interactions between pairs of actions to be taken by different agents. After unsafe interactions between actions have been identified, synchronization actions will be added to the plans to force some agents to suspend activities until other agents complete their actions. It is a powerful technique to increase parallelism in the planning process as well as during execution [101]. But the huge amount of local plans to be exchanged between agents may place a heavy burden on the multi-agent system.

Instead of inconsistencies and conflicting actions, an agent's local plans are often dependent upon the plans of other agents due to the existence of *global constraints*. It requires agents to search through a larger space of plans rather than each proposing a single plan. A common technique to achieve this is called *iterative plan formation*. The hierarchical structures of plans have been utilized to manage interactions at different abstraction levels. Every agent proposes a set of feasible plans [107]. A *search* process is taken to resolve conflicts and dependencies at proper abstraction levels. The planning process keeps progressing until a refined

subset of proposed plans has been identified as the multi-agent plan [109]. Often, this process is mediated by agent negotiation techniques. For example, agents can negotiate to determine which agent should impose further restrictions on its actions in order to avoid conflicts. They may also negotiate to identify potential cooperations so as to improve their mutual benefits. To better facilitate plan execution, agents may hold alternative plans in respond to possible contingencies that arise during execution. They may also restrict their interactions through carefully defined social laws such that adjusting local plans might not cause so much effort [252]. More generally, however, planning and plan execution are interleaved. Coordination is an ongoing process that cannot be performed once for all [100].

An example of the distributed multi-agent planning approach is Lesser and Corkill's *functionally accurate and cooperative* (FA/C) technique [179]. Loosely-coupled agents exchange their local plans to form high-level plans. These plans will undergo a series of refinement until some global complete plan has been identified. Inconsistencies are solved locally among affected agents. FA/C has been applied in Lesser and Corkill's Distributed Vehicle Monitoring Test (DVMT), a system for testing coordination strategies [176]. The testbed is inherently data-driven. The performance is measured as the time used for processing information in order to find the paths of vehicles.

To coordinate agents in DVMT, Durfee developed a planning approach known as *Partial Global Planning* (PGP) [100, 103]. In PGP, every agent can represent and reason about the actions and interactions for groups of agents and how they affect local activities. These representations are called *partial global plans*, which are frame-like structures that represent coordinated actions in terms of goals, actions, interactions, and relationships. Agents use their organizational knowledge to decide how and when they should use PGPs to plan. The organization of the multi-agent system is broken into two parts. The domain-level organization specifies agents' general, long-term problem solving roles and capabilities. The meta-level organization, on the other hand, indicates the coordination roles of agents. With its PGPs, an agent can locally hypothesize the potential interactions between agents. Following the meta-level organization, agents may further exchange PGPs to form larger PGPs and obtain a more complete view of the group activity. This will help them achieve coordination as they can refine their local plans in a more coordinated manner. The PGP approach has been very successful and attracted many research attentions. Based on PGP, Decker has developed the Generalized Partial Global Planning (GPGP) algorithm, which is a set of protocols that agents use in a wider range of applications in addition to DVMT [90, 91].

Partial Global Plannning drives agents' local activities by constantly refining their local plans. This refinement strategy is exploited later by another planning algoirthm termed Fast Forward (FF) [137]. The planning process starts with the initial state and an empty sequence of actions. Repeatedly, the sequence is extended with actions, always adding to the end of the sequence. For each of the possible extensions of the sequence, a heuristic value is calculated. The first of the possible extensions that leads to a state with a lower heuristic value than the current state is chosen.

Kreifelt and von Martial [166] proposed another approach for multi-agent planning. Agents perform distributed planning through a two-stage process. In the first stage, agents plan their activities separately; while in the second stage, they coordinate their local plans and identify a multi-agent plan. Except of the formal definition of an interaction protocol, this approach does not deal with the issue of how agents can achieve consensus through negotiation. Although the research on multi-agent planning has a long and successful history, it is still an active area with many recent research progresses [87, 92, 267, 268].

Comments:

The research on multi-agent planning intends to solve many common coordination problems. The plans are utilized as general artifacts for (1) detecting and avoiding inconsistent or conflicting actions; (2) regulating the share of resources; and (3) managing the sharing of labor towards the planned goal. Multi-agent planning requires agents to share and process a huge amount of information. It may consume a substantial amount of computation and communication resources, especially when the application environment is complex. The centralized planning approach is essentially a task decomposition and passing technique. The objective is to find, among all the possible plans that accomplish a goal, the plan that can be decomposed and distributed more effectively. But since the multi-agent plan is to be synthesized by a designated coordinator agent, it is not certain whether the most decomposable plan can be actually allocated to other agents. The centralized approach also presumes that at least the coordinator agent should have a global view of the entire system. In many domains, this is an unrealistic assumption. It also contradicts with the basic design principles of the multi-agent paradigm, for instance, distributed control, concurrency, robustness, and minimal bottleneck. When the planning activity is distributed among all the participating agents, as in the distributed multi-agent planning approach, the situation becomes even more complex. In the worst cases, each agent will spend equivalent efforts as the coordinator agent in order to obtain

a global system view.

The communication infrastructure can also have a big impact on the degree to which plans can be decomposed and distributed. Practically, some minimal subplan size exists. It does not make sense to decompose a multi-agent plan to subplans with a size smaller than that. In loosely-coupled networks, this implies that agents should each accomplish larger tasks; while in tightly-coupled networks, the degree of decomposition can be increased to improve concurrency. Despite of the computational complexities, flexible coordination behaviors can be achieved through multi-agent planning. But the decision procedures used to identify these local or multi-agent plans remain fixed [106]. To further enhance the flexibility of these approaches, learning mechanisms can be incorporated into the planning processes. For instance, agents can create situation-specific control rules to help them select plans or scheduling strategies according to the past system feedback [258].

Coordination through negotiation

From a intra-agent perspective, negotiation is a natural mechanism for agents to resolve their differences by changing their viewpoints and finding mutually acceptable agreements. Most coordination schemes involves some sort of negotiation. The reason for its popularity is many-sided:

- Among the many possible mechanisms, negotiation is the most fundamental and powerful one for managing intra-agent dependencies at run-time [151].
- Negotiation can be applied to situations when agents are either self-interested or cooperative. It serves as a good framework that helps us understand and analyze the various interactions among agents.
- Coordination techniques based on negotiation are inherently distributed. They are flexible, robust to unexpected changes, and not restricted to any particular forms.

Negotiation is used to modify agents' local decision-making procedures, and identify situations where certain agent interactions are possible. Modification and identification trigger the process of negotiation so that a common decision can be reached [197]. The literature abounds as negotiation is considered as a key issue in DAI. It has been extensively studied in the fields of resource and task allocation [244], conflicts recognition and resolution [4], and dynamic group formation [193].

In the work of Smith [255], negotiation serves as an "*organizing principle*" used to effectively match tasks and problem solvers. Negotiation is a two-way

exchange of information. Both parties evaluate the information from their own perspective and finally reach an agreement based on mutual selection. Despite of the great success of the contract net protocol, this viewpoint on negotiation bears some major shortcomings. Especially, Smith believes that there must be a conflict between agents in order to start the negotiation process. Besides, the possibility of bargaining among agents has also been ignored. It is hence a static view on negotiation and is not flexible enough to express the overall research area. Pruitt describes negotiation as a process by which a joint decision is made by two or more parties. Starting from contradictory demands, an agreement is finally achieved through a process of concession or search for new alternatives [220]. This definition underpins three major research topics of negotiation, which has also been pointed out by Muller in [197]:

- **Negotiation language**: Research is concerned with the communication primitives for negotiation and their formal semantics. This research topic has been covered previously in Subsection 2.3.1.
- **Negotiation model**: general models of the negotiation process are investigated and the global behavior of the negotiation participants are analyzed.
- **Negotiation decision**: Algorithms to (1) compare and analyze the issues being negotiated, (2) find counter-proposals, (3) make arguments, or (3) determine final negotiation results.

As the study of negotiation becomes deeper and broader, several other research issues have received increasing attentions. For example, in order to enhance the system flexibility, various learning mechanisms have been investigated to help agents achieve desirable agreements. The impact of learning in multi-agent negotiation is inarguably an interesting topic, which has attracted intense discussions [43,82,142,297]. Instead of considering each research topic in isolation, it is also interesting to see how these topics are interrelated. For example, Kraus showed that as more restrictions are forced upon the agents by the negotiation protocols, the rationality of these agents decreases [164].

A. Negotiation Model

Many research works on negotiation models have been inspired by game theory [210]. Agents make their negotiation decisions according to some kind of utilities they will gain, which are affected by the utilities and decisions of other agents. Rosenschein is a pioneer in game theory based negotiation analysis [228–230]. He and Zlotkin have identified three distinct domains where negotiation is applicable:

(1) task-oriented domain: agents negotiate to allocate their tasks that will benefit everyone; (2) state-oriented domain: agents negotiate to find actions that change the state of the "world" and serve their goals; and (3) worth-oriented domain: agents negotiate to gain maximum utilities for reaching certain states [230]. Different negotiation strategies have been proposed and examined with respect to these three application domains [298].

In a negotiation process that involves multiple agents, when making a decision, any rational agent A must take into account the probable choices of other agents, who in turn make their decisions based upon A's own choice. This leads to the well-known *outguessing regress* problem where no accurate prediction or confident expectation about the individual choices can be produced. In order to circumvent this problem, the game-theoretic models often take three restrictive assumptions: (1) both the number of agents and their identity are fixed and known to everyone; (2) all agents are fully rational and each of them knows that others are rational; and (3) each agent's risk-taking attitude and expected utilities are fixed and known to every one. These assumptions limit the applicability of game-theoretic based negotiation models [165].

Because of the *outguessing regress* problem, it is difficult to get a general model governing rational choice for multi-agent negotiation. Some game theorists sought to achieve deterministic solutions by introducing the notion that each agent might be able to assign subjective probabilities to the choices of other participants [164]. That is, every agent proceeds in certain subjective fashion to estimate the probable choices of other agents [142]. In essence, this treatment reduces an agent's decision-making problem to a situation that is fundamentally analogous to a game against nature as in a traditional single-agent game.

Game theoretic models focus primarily on the negotiation outcomes. The negotiation processes have to be described through other models. The sequential decision-making paradigm has been used by Zeng and Sycara to model negotiation procedures [297]. They assume that there is a sequence of decision-making points which are dependent upon each other. After making a decision at certain point, agents can update their knowledge through the system feedback so that they may make a more informed decision at the next decision point. This sequential decision-making model provides readily available constructs to describe the iterative nature of agent negotiation. It also supports an open-world environment and enables agents to learn through negotiation. Similar approach is taken by Kraus, who used Rubinstein's protocol of alternating offers to model the negotiation process [164].

Another model of negotiation processes was proposed by Bussmann and Muller [48]. They analyze negotiation from a socio-psychological perspective. Drawing

from Gulliver's eight phases of negotiation, they proposed a cyclic negotiation model. The pure negotiation cycle starts with one, some, or all agents making a proposal to a certain conflict. The negotiation process continues as every agent evaluates the proposals against its own preferences and criticizes them by listing its preferences violated by them. After updating their knowledge about other agents' preferences, these agents resume negotiation by proposing new proposals and starting another negotiation cycle. To emphasize agents' psychological activities, Lander and Lesser modeled the negotiation process as a search procedure [172]. A set of states is defined to represent the negotiation status. These states are changed by applying negotiation search operators to initiate, critique, relax, and remove the proposals being negotiated.

Comments:

This family of research focuses on two main classes of objects for modeling: the *negotiation procedures* and the *negotiation outcomes*. Models on *negotiation procedures* aim at abstracting the negotiation forms from a social perspective. On the other hand, the research on negotiation outcomes concentrates more on the rational choices for multi-agent negotiation from a psychological perspective.

There are some significant problems associated with game theoretic models of multi-agent negotiation:

- Game theory assumes that it is possible to characterize an agent's preferences with respect to outcomes. Humans, however, find it extremely difficult to consistently define their preferences over outcomes.
- Game theory has failed to generate a *general* model governing rational choice in interdependent situations [296]. Instead, the discipline has produced a number of highly specialized models that are applicable to specific types of decision-making situations [298].
- Game theoretic models assume perfect computational rationality, meaning that little computation is required to find mutually acceptable solutions within a feasible range of outcomes.

For certain restricted negotiation situations, game theory is able to provide a theoretically sound mathematical solution and winning strategy. The existence of solutions, however, does not guarantee that agents can find the solution. For example, in the data allocation scenario presented by Kraus, the task of finding an optimal comprise among negotiating agents is NP-hard [164]. Game theory also takes a rather static view on agent negotiation

by defining equilibrium points [12]. These equilibrium points, such as the *Nash equilibrium*, restrain agents to pre-defined results, although the whole negotiation process might be fairly complex and critical.

Certain process models, such as Bussmann and Muller's work [48], can be understood as bridges that link practical applications and game theoretic models. They help players of a game locate approximate solution strategies according to principles of bounded rationality by utilizing heuristic search, heuristic evaluation, or even learning technologies. More or less, they sacrifice the flexibility of the coordination process in order to make agent interactions computationally manageable. For example, Zeng and Sycara assumes that agents are only allowed to make proposals and counter-proposals at each decision point [297]. No other techniques, such as making arguments, are used to further modify or enhance the negotiation process. Even though each agent is able to make their local decisions freely, agent coordination in these models are believed as fixed processes or processes with pre-defined branches.

B. Negotiation Techniques

Facing a multi-agent encounter, agents need specific negotiation techniques to solve realistic problems. The common negotiation topics are tasks, plans, resources, intentions, preferences, or goals of agents. Agents can be either cooperative or competitive (self-interested) during negotiation. Complex structures are also allowed such that multiple agents may form a coalition or establish a separate agreement.

Drawing inspiration from competition in human societies, researchers have designed negotiating multi-agent systems based on the supply and demand formalism. The most successful one is the contract net protocol [255]. In this approach, agents assume two roles: (1) a *manager* who breaks a problem into sub-problems and searches for contractors to do them; and (2) a *contractor* who does sub-tasks awarded to it. Contractors may recursively become managers and further decompose sub-tasks and sub-contract them to other agents. This is a completely distributed scheme where managers locate contractors through a bidding process. The advantages of the contract net protocol include dynamic task allocation via self-bidding, open system support, natural load-balancing capabilities, and reliability through distributed control and failure recovery. It also suffers several drawbacks, including the fact that it does not presume agents with contradictory demands [241].

Many researchers have enhanced the contract net protocol to solve problems of particular domains. For example, Conry *et. al.* generalized the approach to form a multistage negotiation protocol [80]. It is designed for a complex communication system involving transmission path restoral. In this system, network sites are partitioned into several geographical regions with a single cite in each region being designated as a control cite. Each control cite has responsibility for communication system monitoring and control within its region. Because the regions are inter-connected with each other, each decision made by one control cite could have impact on the operations of other regions. The multistage negotiation protocol seeks to solve this problem by extending the contract net protocol. It enables agents to iteratively exchange their assessments about the impact of their own choices. This strategy is similar in character to the FA/C paradigm [179] and contributes to the incremental construction of negotiation agreements, which are globally consistent. To move even further, Sandholm extended the contract net protocol to cover the situation where agents are self-interested and deceitful [242].

It appears that almost every form of human interaction involves some degree of explicit or implicit negotiation [197]. It is hence not very surprising that many negotiation researches are drawn from human negotiation strategies. It leads to the use of miscellaneous AI techniques including case based reasoning, planning, constraint-directed search and so on. As Jennings *et. al.* pointed out, the key advantages of these heuristic approaches are [151]:

- Negotiation techniques are based upon realistic assumptions. They provide a more suitable basis for automation and can therefore be used in a wider variety of application domains.
- The system designers, who are not wedded to game theory, can use alternative and less constrained models of rationality to develop practical negotiation strategies.

The PERSUADER negotiation system proposed by Sycara handles multi-agent, multi-issue, single or repeated negotiation based on an integration of case-based reasoning and multi-attribute utility theory [261]. This technique is able to narrow agents' differences through exchanging proposals and counter-proposals. It allows agents to reason about and even modify other agents' beliefs. Given the input of a set of potentially conflicting goals, the final output is either an agreement or an indication of the failure of negotiation. Agents mainly perform three tasks iteratively to reach the final output: (1) propose an initial compromise; (2) repair and improve a rejected compromise; and (3) change an adversary's merits of a proposal. The major contribution of PERSUADER involves the use of case-based reasoning to propose new proposals, the use of multi-attribute utility theory

to model agents' preferences, and the use of argumentation to affect other agents' beliefs. If agents prefer arguments that are more likely to reach an agreement, it is possible to prove that argumentation may expedite the negotiation process and save agents' coordination efforts [272].

Sathi and Fox view agent negotiation as a constraint-directed search [244]. The work is based on the observation that the use of different negotiation strategies by negotiators is dependent upon the negotiation space and constraint structures. In situations where constraints affect only individual offerings, a distributed negotiation in the form of a market is suffice. But if most of the constraints are conditional upon multiple offerings, mediator-driven negotiation seems more appropriate. Agent negotiation is supported by designated mediators that take the responsibility of constructing transactions and cascades. As negotiation is viewed as a constraint-directed search, three operators have been proposed to change the position in the problem space, which are inspired by human problem solving practices. These operators are responsible for constructing and modifying transactions and cascades. It has been shown that this search based negotiation approach outperforms human experts in a computer allocation scenario [244].

Alder *et. al.* examined negotiated agreements and the method of conflict detection and resolution using planning techniques [6]. Because agents need information from others to function effectively and efficiently, the planning process is intertwined with the negotiation process. Agents use their planning knowledge to evaluate the proposals and counter-proposals made by other agents and in turn make their own proposals.

Due to the limited perception capability and bounded rationality, agents are often uncertain about themselves, other agents, and the environment. Techniques that tackle uncertainties have been extensively explored for agent negotiation. For example, fuzzy logic has been used to model agents' preferences. Faratin *et. al.* used fuzzy similarity to compute tradeoffs among multiple attributes during bilateral negotiation [110]. Kowalczyk models the multi-issue negotiation as a fuzzy constraint satisfaction problem [163]. Agents perform negotiation on individual solutions once at a time. Offers are evaluated and constraints are relaxed during the negotiation process in order to find agreements for both parties. Luo *et. al* proposed a fuzzy constraint based framework for bilateral multi-issue negotiation in competitive trading environments [186]. The framework is expressed via two knowledge models, one for seller agents and one for buyer agents. The buyer and seller agents exchange offers and counter-offers with additional constraints revealed or existing constraints relaxed. Finally, a negotiation solution is found if there is one.

In addition to modeling agents' preferences, fuzzy logic techniques have been used to solve bidding problems in a multi-agent double auction environment [133]. Agents use predefined rules to decide the price of their next bids based on the history and current bidding conditions. Empirical results demonstrate the advantages of this approach to selected heuristic decision-making strategies. Except fuzzy logic, statistic techniques have been used to perform multi-agent reasoning under uncertainty for bilateral negotiations. In the work of Gimenez-Funes *et. al.*, uncertainties due to the lack of knowledge about other agents' behaviors are modeled through possibility distributions [124]. Based on the information from a case base of previous negotiation situations, the possibility distribution is generated by choosing the most similar situation and price from the case base.

Without using fixed decision policies, learning mechanisms have been utilized to improve the flexibility of the negotiation/coordination process. Zeng and Sycara used Bayesian learning techniques to estimate the probability distribution of the opponent's reservation price [297]. Bayesian learning is also used by Bui *et. al.* to learn the preferences of other agents [43]. Their approach has been successfully applied to a distributed meeting scheduling problem. Ho and Kamel proposed a multi-agent probabilistic hill-climbing method to help agents learn their negotiation strategies in cooperative settings [136]. Abul *et. al.* embedded reinforcement learning mechanisms into agents' decision process to help them take coordinated actions [3]. Using a training process, Ontanon and Plaza imbue their agents with decision trees, which help these agents to determine when they should collaborate with other agents during negotiation [209]. In a multi-agent meeting scheduling problem where distributed agents negotiatte meeting times on behalf of their users, Crawford and Veloso utilizes the *playbook* approach [38] for team plan selection to help agents learn the desirable negotiation strategies [82].

Comments:

> Four issues of concern that underpin agent negotiation techniques have been identified by Sycara, which are (1) ensuring network coherence; (2) facilitating task and resource allocation among agents; (3) recognizing and resolving disparities in agents' goals and viewpoints; and (4) determining organizational structures [261].
>
> Market like negotiation mechanisms, such as the contract net protocol, provide a natural way of distributing tasks. They are most appropriate when (1) the application task has a well-defined hierarchical structure; (2) the problem has a coarse-grained decomposition; and (3) there is minimal coupling among subtasks [144]. These approaches, however, failed to presume agents

with contradictory demands. The conflicts between agents' goals are neither detected nor resolved (no exchange of proposals and counter-proposals or argumentation). Agents are supposed to be passive and benevolent. In many real-world problems, this assumption is unrealistic. Meanwhile, protocols like the contract net can also be very communication intensive as agents keep on sub-contracting tasks with complex structures. Despite of the potential drawbacks, the contract net protocol does not control the internal decision-making procedures of each agent. Agents can either use fixed decision rules or even establish their decision policies through learning.

For planning based negotiation techniques, the problem of how agents can really achieve consensus is often vaguely described. Further clarification is needed to formalize agents' planning and negotiation behaviors. In essence, these negotiation techniques are analogous to multi-agent planning methods and share the same advantages and drawbacks. Unlike the contract net protocol, agents have to rely on their planning knowledge to make decisions. The coordination processes are flexible in that no fixed procedures or branches should be followed. But for certain applications, they are not flexible enough since agents may not adapt their planning activities with respect to the changing environment through learning mechanisms.

Most of AI-based negotiation techniques seem to be either problem dependent or used simply because their designers have special experience in using them. Although problem dependent issues should not be ignored, there lacks clear methodology of applying these negotiation techniques in general application domains. Extensibility holds another concern. For example, when a huge amount of buyers and sellers with intricate relations are present, the constraint-directed search supported by Sathi and Fox's negotiation system may consume tremendous computation resources before any agreement can be finally reached [244]. Similar problem is faced by other negotiation techniques, such as the one proposed by Zeng and Sycara [297]. When a lot of issues are possible to be negotiated and agents' preferences over those issues are interdependent, it can be very difficult for one agent to learn other agents' preferences, especially when negotiation happens occasionally. Last but not least, the concept of agents' preferences has won great popularity in agent negotiation research. However, not so many techniques have treated this concept rigorously: how these preferences should be computed and what their values stand for psychologically.

Coordination through norms and social laws

One of the most important techniques we use to coordinate our activities during everyday life is to follow *norms* or *social laws* [181]. Norms are established patterns of behaviors. They are introduced to strike a balance between individual autonomous and the goal of the multi-agent system. Agents' decision-making problems are simplified since norms define the course of actions to be followed in certain situations. One key issue of using norms is to find the most effective method by which agents can establish norms that benefit the whole society. Two main approaches are discernible: *offline design* and *spontaneous emergence* [292].

For the *offline design* approach, norms are determined offline and hardwired into every agent. Several studies of the offline design with a particular interest in the analysis of the computational complexities have been reported in the literature [252,253]. Shoham and Tennenholtz define norms as a set of constraints. An agent is said to be legal if it never attempts to perform an action that will break the constraints imposed by the norms. An agent is always allowed to visit a group of states termed *focal states*. A norm is *useful* if it does not prevent the agent from reaching any of the focal states. As proved in [252, 253], the problem of finding useful norms for a given application domain is NP-hard. It poses a technical challenge on the practical usability of the offline design approach.

Spontaneous emergence approach investigates how norms can emerge in a society of agents. Norms are global knowledge since all agents use them. But each agent must decide on which norm to use based solely upon its own experiences. The major research issue hereby is to design strategies that will lead each agent to agree upon the use of certain norms by using only the local information accessible to it. The possibility that such strategies exist is investigated in several works [161,251,278]. Shoham and Tennenholtz considered the emergence of norms in a *tee shirt game* [251], where each agent decides locally the color of the shirt it wears. Four different strategies have been used to establish norms: (1) simple majority; (2) simple majority with agent types; (3) simple majority with communication on success; and (4) highest cumulative reward. Shoham and Tennenholtz find that the strategy of highest cumulative reward enjoys the highest *efficiency of convergence*. It was shown that for any ϵ, $0 < \epsilon \leq 1$, there exists a value n such that all agents will agree on a certain norm in n rounds with a probability $1 - \epsilon$, when the strategy of highest cumulative reward is used [254].

Comments:

This family of research seeks to simplify agents' decision-making by imposing norms and social laws on the multi-agent system. The main research question is on the methods that will lead to the establishment of beneficial norms. The offline design approach gives system designers a higher degree of control over system functionalities. It is both simple to implement and understand. But the more complex a system becomes, the more difficult it is for system designers to predict the impacts of their norms and therefore identify effective norms that satisfy on-going system requirements. Furthermore, norms reduce the flexibility of agents' decision-making processes such that, in certain situations, only fixed coordination processes will be followed.

The spontaneous emergence approach improves the flexibility of the coordination process since agents have to determine their norms jointly according to the current system status. Nevertheless, no clear concept of usefulness applies to the norms identified by agents. For example, to decide on the color of tee shirts has nothing to do with the agents' goals or the global system performance. There lacks clear reason that explains why such a norm on colors will need to be established. The *stability* of norms serve as another issue that requires further study. Specifically, we expect that the identified norms will not immediately fall apart so that agents' efforts are not wasted without getting any results. On the other hand, agents must avoid being over-committed to any particular norm and should be open to new ideas. Mechanisms that properly balance these two criteria stand for a interesting topic for future research.

Coordination through mutual modeling and joint intentions

When we work together, it is a common practice for us to *put ourselves in the place of others* when determining our own actions. Inspired by this fact, Genesereth *et. al.* first articulated that agent coordination can be achieved through *mutual modeling* [121]. The idea is that if you share a common view of the scenario with other agents, you can perform a timely analysis to predict what is the rational choice of each agent and then decide your own actions accordingly.

One technique that coordinates agents through mutual modeling was proposed by Gasser *et. al.* in the form of the MACE system [116]. MACE contains five main components: (1) a collection of application agents; (2) a collection of predefined system agents; (3) a collection of system facilities; (4) a description database; and (5) a group of kernels (one per physical machine). Each agent in MACE can be described in terms of three aspects: (1) its knowledge; (2) its perception of the environment; and (3) the actions it performs [116]. Agents have two kinds of knowledge: domain knowledge and *acquaintance* knowledge. The *acquaintance*

knowledge is the knowledge about other agents, such as their skills, goals, or plans. They are obtained through sending and receiving messages. Based on its acquaintance knowledge, an agent may be able to identify potential conflicts and inconsistencies and modify its own behaviors appropriately. Similar techniques have also been applied by Gasser *et. al.* to a *predator and prey* game [24], where the organization knowledge (or knowledge about other agents) is described through a group of settled and unsettled questions [117].

In addition to mutual modeling, when humans cooperate as a team, mental states that are closely related to *intentions* appear to play a similarly important role [180]. Intentions provide both the predictability and the flexibility that are necessary for agents to coordinate in a changing environment. Two types of intentions are discernible: individual intentions and collective intentions. In addition to pursuing its own goal through individual intentions, as Levesque *et. al.* pointed out, an agent who has collective intentions will show some sort of social *responsibility* towards other agents [180]. If it discovers that the team effort is worthless, it should at least attempt to make other agents aware of this.

Inspired by the work of Levesque *et. al.*, Jennings identified two important concepts of teamwork: *commitments* and *conventions* [147,148]. A *commitment* is a pledge or a promise of performing a course of actions. Commitment is *persistent*. Having adopted a commitment, an agent will not drop it until, for some reason, the commitment becomes redundant. A *convention* is a means of monitoring a commitment. It specifies the conditions under which a commitment should be abandoned and how an agent should behave both locally and towards other agents. Jennings further proposed that when a group of agents were engaged in a cooperative activity, they must have a joint commitment to the overall aim, as well as their individual commitments to the specific tasks that have been assigned to them [148]. The state of this joint commitment is distributed among the team members. At the time when it is going to be dropped, agents should behave towards its fellow team members according to the corresponding conventions.

This concept of commitments and conventions has been explored in industrial control systems such as ARCHON and GRATE [146, 150]. Agents take a layered architecture in ARCHON, which contains three main control modules: the *cooperation* module, the *situation assessment* module, and the *communication manager*. The cooperation and situation assessment modules are responsible for controlling agent's coordination activities. They help agents identify, establish, and eventually terminate cooperation through four stages [293]:

- Recognition: some agent in a multi-agent system recognizes the potential for cooperation in order to achieve a certain goal.

- Team formation: after recognizing the potential for cooperation, agents seek help from other agents to form a team (a collective deliberation stage).
- Plan formation: under the joint commitment of cooperation, agents decide how to actually achieve the desired goals by forming an agreement or a plan.
- Team action: the newly agreed plan of joint actions is executed by team members. When contingencies occur, certain conventions will be followed by each agent to replan or terminate cooperation.

Another coordination framework similar to ARCHON was developed by Tambe and was called *Steam* [265]. The cooperation component of Steam is encoded in about 300 domain-independent rules. It allows the construction of complex coordination processes, such as forming hierarchical team structures. The usefulness of Steam has been investigated in military mission simulations and the RoboCup robotic soccer domain.

2.3.4 Bottom-up Coordination Technologies

Bottom-up coordination approach attempts to build less sophisticated agents and see what emerges when they are put together into a group. Agents can be either cooperative or competitive. Communication between agent is either allowed or prohibited. The whole multi-agent system may also be populated with either homogeneous or heterogeneous agents [257]. Most of the technologies follow a behavior-based methodology [41]. The underlying philosophy is that intelligent behavior is not disembodied, such as the symbolic/logical approach to building agents, but is a product of the interaction between agents and their environment.

The flexibility of bottom-up coordination technologies vary from simple reactive-based agent systems to complex agent systems where learning mechanisms have been extensively explored. Due to the highly complicated dynamics of multi-agent systems, there lacks systematic approach of designing agents' reaction rules. In general, the ability to orchestrate good global performance via local interactions remains as a significant and ill-understood challenge. As a result, in many situations, the desirable system behavior is better to be viewed as a surprise rather than as an expected result. Research on bottom-up coordination technologies is still at its beginning stage. More works are needed in the future in order to put them into practical use. In the remaining of this subsection, several technologies will be introduced to give a glance of this booming research field.

In the simplest cases, agents retrieve predefined behaviors similar to reflexes without maintaining any internal state. Balch and Arkin used reactive agents to study formation maintenance for moving agents [14]. The agents' goal is to move

together in a well-defined formation such as a diamond or a column. Periodically, they will come across obstacles, which will disturb the current formation they maintained. After passing the obstacles, all agents use reactive signals convert from their sensory data to control their motions and re-establish the formation disturbed. Reactive-based approach is effective in this case since the coordination objective is to maintain certain formations, which may not depend upon the system history. Kephart *et. al.* considered a case of multiple agents sharing a set of resources [160]. Agents have only limited and delayed perceptions of their environment. Using this information, they decide individually which resources to use in order to receive more rewards using reaction rules. Kephart *et. al.* shows that (1) imperfect knowledge suppresses oscillatory behavior at the expense of reducing performance; (2) enhancing the decision-making abilities of some of the individual agents can either improve or severely degrade the overall system performance; and (3) the system can remain in nonoptimal metastable states for extremely long periods of time before escaping to the global optimal state.

Unlike reactive style of coordination, deliberative agents behave more like they are thinking by maintaining a group of internal states and continually adjusting their decision policies. There is no clear line between reactive and deliberative agents. Some agents simply mix reactive and deliberative behaviors. One example technique is termed *reactive deliberation*, which is proposed by Sahota [240]. Reactive deliberation mixes reactive and deliberation behaviors. Based on its internal state, an agent reasons about which reactive behavior to follow under certain application constraints. It has been successfully applied to the first robotic soccer platform [17]. To achieve further flexibility during agent coordination, learning technologies can be explored by each agent such that a collective adaptive behavior emerges for the group of agents as a whole.

Maes and Brooks investigated a distributed learning algorithm that learns to coordinate the behaviors of the six legs of an insect-like robot [187]. Each of the six legs is considered as an agent. Each agent observes individually which legs are currently touching the ground. It must learn the conditions under which it should swing its leg forward. All agents learn concurrently and receive the same global reinforcement signals. Positive feedback results from forward motion; while negative feedback comes when the insect has fallen. Maes and Brooks showed that the insect eventually learns a tripod gait, where it lifts up three of its legs at a time. In a similar fashion, Humphrys presented a behavior-based algorithm where each behavior of a robot is considered as an agent competing for the right to control the robot [145]. Agents perform locally Q-learning algorithms with their own reward functions. Every agent suggests an action with some weight based on its Q-values and the robot executes the action with the highest weight.

To further improve the system performance, reward structures are evolved using genetic algorithms according to certain fitness measure. Although Humphrys showed that the algorithm can help the robot achieve a high fitness, a higher fitness may be reached by the robot when it learns through a centralized learning process.

A similar scenario with more deliberative agents is explored by Mataric [191]. In this case, agents use Q-learning to learn behaviors including foraging, homing, and flocking. Flocking behavior is obtained by having each agent sum the outputs of its individual avoidance, aggregation, and wandering behaviors. Foraging is achieved through a more complex combination of behaviors that are sensitive to environmental conditions. In order to speed up the learning process, certain domain knowledge is used to reduce the complexity of the learning algorithms.

Recent research has witnessed a booming application of learning algorithms (e.g. Q-learning) under the bottom-up coordination approach [3, 39, 112, 155, 192, 214, 274, 275]. For example, in [275], Touzet applied lazy Q-learning to a certain class of applications called cooperative multi-robot observation of simple moving targets (CMOMMT). The research focuses on studying the efficiency of the Pessimistic Algorithm in its task of inducing learning of cooperation [275]. A more detailed introduction of serveral other learning based coordination approaches can be found in Section 4.8 of Chapter 4.

Instead of being inherently cooperative (have identical/compatible goals), a group of selfish agents can also learn to be cooperative. For example, in the Prisoner's Dilemma, agents are selfish but not inherently competitive [135]. They can benefit from cooperation without following the Nash equilibrium which is to confess. Axelrod uses the Prisoner's Dilemma to study how cooperation can develop implicitly [11]. He showed that in repeated games, there is no best strategy independent of the strategy used by other agents. Axelrod conducted a *evolutionary tournament*, where the number of offsprings of each strategy was proportional to the score in its last tournament. It has been found that strategies that were successful would proliferate at first, but later shrink as new strategies appear. This process may possibly lead to an escalating "*arm race*" with no end. Agents continually adapt to each other in a more and more specialized way. It has been demonstrated by Sandholm and Crites that a learning agent is able to perform optimally against a fixed strategy [243]. But when all agents are learning, there is no stable solution.

Another important issue of concurrently learning agents is the credit-assignment problem. When the performance of an agent improves, it is not clear whether it is due to the improvement of its own decision strategy or due to the change of other agents' strategies. One way to handle this problem is to fix one agent while

evolving the other agent and then switch (in the case of two learning agents). For example, Rosin and Belew uses this technique and genetic algorithms to evolve agents in the Prisoner's Dilemma [231]. Their approach, however, encourages the arm race more than ever. The society of agents may therefore never fixed on any strategies.

2.4 Summary

This chapter gives an overview of the diverse literature on coordination techniques in multi-agent systems. Without proper coordination, the multi-agent system paradigm simply looses most of its benefits. As we have described in this chapter, various coordination approaches have their relative advantages and drawbacks. No universally best technique exists for all kinds of situations. In general, theoretical work produces good results in well-constrained environments, but many of its underpinning assumptions are violated in real-world applications. On the other hand, implementation-oriented work operates well in specific domains but suffers from the lack of a grounding and rigorous evaluation. In the future, extensive theoretical works are required to develop models that address a broad range of issues essential to agent coordination research. This should lead to well-specified implementations of coordination algorithms which have clearly delimited ranges of applicability. More systematic evaluation of these coordination techniques can therefore be performed so that system designers can make their decisions based on empirical evidence instead of on *ad hoc* assumptions.

To pave the way for the discussions in the following chapters and to conclude this Chapter, The following summarizes the common coordination problems (in the area of sujective coordination research) that are closely related to the research reported in this thesis.

- **Coordinated resource sharing**: agents need to coordinate the sharing of their resources in order to achieve their goals due to the incompetence of any single agent or the global constraints. In Chapter 3, the FSTS model is proposed to model agents' resources. In our research, agents do not share their resources directly. Instead, they depend on the resources to perform and share their actions.

- **Coordinated labor sharing**: similar with the resource sharing, in many situations, agents need to coordinate their sharing of labor. In this thesis, labor sharing is achieved through reinforcement learning methods in Chapter 4 and through dynamic optimization methods in Chapter 6.

- **Inconsistency management**: in order to organize their local behaviors

well, it is often necessary for agents to recognize and resolve disparities in their subjective belief, desires, and intentions. A highly abstract form would also involve the identification and resolution of conflicting plans and goals. Aiming at improving the learning performance, Chapter 5 proposes a protocol that ensures a consistent local knowledge of each agent.

- **Organization control**: for many cases, organizational knowledge is important for agents to decide (1) when they should coordinate; (2) with whom they should coordinate; and (3) how to resolve conflicts and achieve coordinated decisions. This thesis assumes that all agents are cooperative under a pre-defined organizational structure. However, agents still need to adjust their local behavior such that the whole organization can run smoothly.

Chapter 3

Fuzzy Subjective Task Structure Model

3.1 Introduction

Managing the interdependencies among agents is often a highly complicated task and challenges the successful application of the multi-agent system paradigm. In a task-oriented domain, this challenge results from the intricate relations among agents and tasks, or the uncertainties agents have about themselves, the relations with other agents, and the environment. This motivates our research in constructing a coordination model that deals with agents' beliefs, intentions, and uncertainties in a consistent manner. Fuzzy logic techniques have been explored in the model to tackle the inherent uncertainties within agents' local beliefs. Agent coordination is regarded as a series of decision-making processes in uncertain environments. This view is inspired by the extensive research in AI in Decision Theoretical Planning (DTP). DTP is considered as a general paradigm for planning and decision-making problems in uncertain settings [36], and roots in the Markov Decision Process (MDP) [23, 221, 288]. In this chapter, the Fuzzy Subjective Task Structure model (FSTS) is presented first. With FSTS, an agent coordination problem is further showed to be a DTP problem where the Markov property holds. The problem is then solved in the next chapter by the reinforcement learning techniques.

3.2 Related Work

In the literature, the modeling of the task-oriented domain is described in [90] where a model named **TÆMS** is proposed. **TÆMS** has three levels: *generative*, *objective*, and *subjective*. Only the *subjective* level describes agents' local view of the objective problem-solving situation. The construction of the *subjective* level model is described as a *subjective mapping* that maps from the elements in the *objective* model to the agent's subjective view of those elements. In comparison,

the FSTS model described in this chapter focuses mainly on agents' subjective beliefs where the uncertainties about the application domain have been explicitly presented.

In essence, the FSTS model establishes the link between agent coordination and MDP, through which reinforcement learning algorithms are eligible to be applied. Generally, MDP lays the foundation for much of the work on stochastic planning. The rapid progress of this area impels researchers to explore and extend the basic model in order to cater for the multi-agent settings. One example of this is the Multi-agent Markov Decision Process (MMDP) proposed by Boutilier [36]. The MMDP is a straightforward extension of the MDP by factoring the action space into actions for multiple agents. The FSTS model follows basically the same approach. Nevertheless, in order to tackle the inherent complexities of MMDP, FSTS focuses on a refined category of applications termed task-oriented domains. The structures in tasks are explicitly modeled in FSTS and utilized by the reinforcement learning algorithms described in Chapter 4.

Other researchers have also taken a similar view as MMDP. Guestrin *et. al.* concentrates on interactions between agents through the reward function [130]. This approach is also taken in [19, 123]. Without assuming that the global system state is fully observable, there are other models that are more of an extension of Partially Observable Markov Decision Processes (POMDPs) to multi-agent systems. Representative research includes the Partially Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin *et. al.* [218] and the Decentralized Markov Decision Process (DEC-MDP and DEC-POMDP) proposed by Bernstein *et. al.* [27].

Since partial observability will significantly increase the problem complexity, whenever possible agent communication should be considered as an important solution for maintaining a consistent global system view. This has encouraged many researchers to explicitly include agent communication into a MDP framework. Examples involve the Communicative Multiagent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe [222] and the DEC-POMDP with Communication (DEC-POMDP-COM) proposed by Goldman and Zilberstein [128]. The FSTS model does not assume a fully observable system state since it models only agents' subjective beliefs and intentions. However, in order to improve the learning performance, a communication protocol is proposed and utilized to ensure the availability of the latest global knowledge. The details can be found in Chapter 5.

To extend the modeling capabilities of MDP, recent research has also explored the game theory to model agent interactions. This includes the Markov game [156] and extended stochastic game [142, 218]. The FSTS model does not rely on the

game theory. The payoff of each agent is defined through the reward function which measures the task completing performance.

3.3 The FSTS Model

Agent coordination models provide a conceptual foundation for describing and analyzing coordination problems in multi-agent systems. The model must be general enough to allow it applicable to various application domains and to be used in various multi-agent systems. On the other hand, it must be specific enough to provide specific modeling elements directly usable in particular application domains. The FSTS model proposed in this chapter attempts to strike a balance between these two but is designed for the task-oriented domains. As many application can be identified as *task-oriented* [230], the FSTS model is still quite general [58, 60].

A task-oriented environment is assumed to be *cooperative*. Agents share the *common goals* when completing a series of *tasks*. Informally, a task is a set of domain operations to be performed under various system constraints. The set of domain operations and system constraints constitute a task's *internal structure*, which determines how the task is actually performed. Assume that both the number of tasks and the number of agents are finite. In a cooperative multiagent system, multiple agents, denoted by $\mathcal{A} = \{A_1, A_2, ..., A_n\}$, coordinate their behaviors in order to fulfill their tasks, denoted by $\mathcal{T} = \{T_1, T_2, ..., T_m\}$. Since each task $T_k \in \mathcal{T}$ can be completed by performing a finite number of domain operations, the system will cease running definitely. Specifically, the task-oriented environment considered in this paper is of an *episodic nature*.

Agent coordination involves determining (1) a set of domain operations, and (2) the exact time to perform each operation. Every domain operation is modeled as a distinct domain *method* or *method* for short, M. Generally, a task can be completed through performing different sets of methods.

A *meta-method*, denoted as $\overline{M}$, is defined as a fuzzy set of methods that satisfy certain criteria (Cr) and are *exchangeable* with each other. It is a more abstract and higher level notion than methods, and allows for a compact representation of tasks. The methods are said to be *exchangeable* if the execution of one method renders other methods inapplicable. In essence, a meta-method is an abstraction of a cluster of exchangeable methods applicable for the fulfillment of a given task. Instead of listing explicitly all of the methods involved, the task can now be described compactly in terms of meta-methods.

The membership degree of any method M with respect to $\overline{M}$, $\mu_{\overline{M}}(M)$, is the *similarity degree* of M. It indicates the degree of satisfaction of M with regard to the criteria Cr (each $\overline{M}$ has a corresponding Cr). The criteria can be

given by users or provided *a priori* by system designers. The criteria and criteria manipulation procedure may be directly coded into agents, enabling the agents to evaluate the similarity degree of any methods at run-time.

The internal structure of a task cannot be fully determined only by a set of meta-methods. Various system constraints exist and they are part of the internal structure of a task. There are two categories of system constraints: *hard constraints* and *soft constraints*. They determine the appropriate methods for a task and their execution order. The FSTS models the constraints as *relations* among meta-methods. Thus, two types of relations, *hard relation* and *soft relation*, are used for describing hard constraints and soft constraints, respectively. Methods and their relations are represented using relation graphs. Formally, a *relation graph* g is a directed graph, and is represented as a tuple of four components $g = (V, E, \epsilon, \beta)$ where

1. V denotes a set of vertices, representing meta-methods;

2. $\epsilon : V \rightarrow L_V$ is the function assigning labels to a vertices; Specifically, ϵ assigns to each vertex u a meta-method $\overline{M}^u$.

3. $E \subseteq V \times V$ denotes a set of edges representing relations, which are either hard relations or soft relations.

   $\beta : E \rightarrow L_E$ is the function assigning a label (a particular relation) to each edge.

A hard relation graph and a soft relation graph is discernible. A hard relation graph represents hard constraints only and a soft relation graph contains merely soft relations. Two classes of *hard relations* are of our major interests. They are *enable relations* and *cancel relations*. The following two rules govern the execution of two domain methods restricted by a hard relation:

Enable rule If the label of the edge (u,v) in a hard relation graph g is of an *enable relation*, then any method $M' \in \overline{M}^v$ becomes performable after executing a method $M \in \overline{M}^u$.

Cancel rule If the label of the edge (u,v) in g is of a *cancel relation*, then any method $m' \in \overline{M}^v$ is canceled by performing (or starting to perform) a method $M \in \overline{M}^u$.

A Cancel rule is illustrated in Figure 3.1 where a cancel relation has been established from meta-method $\overline{M}_1$ to $\overline{M}_2$. Following the Cancel rule, if a method $M \in \overline{M}_1$ has been performed, any attempt to perform a method $M' \in \overline{M}_2$ will definitely induce a failure. The description of hard constraints is usually

imprecise. Thus, the *hard relations* may only be partially tenable. For the cancel relation given in Figure 3.1, equation (3.1) is used to evaluate the degree to which this cancel relation may be violated. The operator $\bigwedge$ is the standard *T-norm operator* [295].

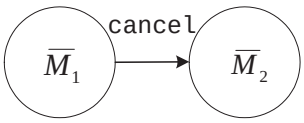

Figure 3.1: The cancel relation between meta-methods $\overline{M}_1$ and $\overline{M}_2$.

$$\mu_{\overline{M}_1}(M) \bigwedge \mu_{\overline{M}_2}(M') \tag{3.1}$$

For a *soft relation graph* g representing *soft constraints*, the function β assigns to each edge of the graph a *soft relation*. In FSTS, we describe a soft relation through a group of *relation parameters*, a mechanism similar to the notion of *non-local effects* [90]. For example, due to the existence of soft relations, the execution of one method may alter the *processing time* and *success rate* of the other method where the *processing time* and *success rate* are two examples of relation parameters. The success rate refers to the probability that the corresponding method can be completed successfully. With these relation parameters, the function β is defined as a mapping of the form β: $V \times V \rightarrow (x, y)$; that is, for any edge (u, v) of the graph g, function β assigns to it a label (x, y) where x and y are two parameters that take real values between -1 and $+1$. As a soft constraint, it states that the execution of a method $M_1 \in \overline{M}^u$ may affect the execution of another method $M_2 \in \overline{M}^v$ through changing M_2's processing time and success rate by a percentage of x and y respectively. The sign of x and y serves as the indication of the influence directions. To characterize the degree to which extent a method M is affected by certain soft constraints, we will follow a simple procedure and utilize another concept termed *method relation graph*. Since this characterization is not unique (not eligible to be included in the FSTS model), its discussion is given in the next chapter where the two reinforcement learning algorithms are presented.

With the notions of methods, meta-methods, and relation graphs, a *task* T is defined as a tuple of two components $T = (\{\overline{M}\}, \{g\})$. $\{\overline{M}\}$ is a set of meta-methods. $\{g\}$ is a group of relation graphs. We assume that for any task T, both sets $\{\overline{M}\}$ and $\{g\}$ are finite.

During the coordination process, a sequence of methods, denoted by Seq_k, that an agent A_k decides to execute is called A_k's *local method sequence*. Formally, Seq_k is a list of *method-time pairs*, (M, t), where M is a domain method chosen by agent A_k, and t is the time when A_k starts to perform M. The local method

sequence represents the decision made by an agent under the assumption that the agent will not execute simultaneously more than one domain methods (the execution periods of two methods overlap).

The *global method sequence*, Seq, is defined as the ordered list of local method sequences of all agents, that is, $Seq = (Seq_1, Seq_2, ..., Seq_n)$. By following Seq, the tasks in $\mathcal{T}$ will undergo a series of changes. A task is said *concluded* if it is unnecessary to perform any domain methods in order to achieve the task. At that time, a *reward*, R, represented by real numbers, is incurred to quantify the degree of satisfaction of the corresponding system goals. A negative R represents a *penalty* [88].

Modelled in FSTS, the coordination is actually a decision-making process by each agent. In the process, each agent makes decision that whether or not it will

- stay idle
- continue a chosen domain method
- select a new method to perform.

The outcome from the decision choices of all agents forms a global method sequence Seq. The performance of such Seq, and further the degree of coordination among agents are to be evaluated from the rewards incurred by following Seq. The objective of agent coordination is in fact to establish a proper "*allocation*" relation between agents and domain methods (to *allocate* domain methods to each agents to perform) in order to obtain a good system performance.

3.4 Agent Coordination as DTP Process

In this section, we provide a formal characterization that the FSTS model of the multiagent coordination in a task-oriented environment actually formulates a Decision-Theoretic Planning (DTP) problem. It allows agents to reason about problems in which actions may have nondeterministic effects and to act rationally to achieve their goals which may not be well defined. The DTP approach is attractive to agent coordination in task-oriented environments which exhibits the following features: (1) as the system constraints may only be partially satisfied, the outcome of performing any method is not deterministic; (2) the reward given to any concluded task is affected by multiple, potentially conflicting system goals.

The formulation of a DTP problem addresses the following four major aspects of the problem.

1. the system states;

2. the decisions to be made by each agent;

3. the stochastic model of system transitions;

4. the quality of the agents' decisions.

The following subsections present a much detailed discussion of these four aspects.

3.4.1 The System State

A system *state* is defined to be a description of the system at a particular point in time. The actual forms of states may vary with different applications. Nevertheless, the state has to capture all the information necessary to the agents' decision-making process. Particularly, we assume that given a global method sequence Seq, the states satisfy the so-called *Markov property* which states that the knowledge of the present state renders information about the past irrelevant to predicting the future system states. In other words, the future state depends *only* on the present state, not on the past (not directly at least).

Let S^t denote the system state at time t and $\mathcal{S}$ denote the set of all system states (state space). Suppose that the system starts running at time t_0. We use a list of time, $(t_0, t_1, t_2, ...)$, where $t_i > t_j$ whenever $i > j$, to denote the specific time sequence at which the system has changed its state, and is determined by the global method sequence Seq. We employ another list termed the "*system history*" to describe the system's behavior under Seq. Specifically, one potential system history under Seq, h, is represented by $h = (S^{t_0}, S^{t_1}, S^{t_2}, ...)$. The set of all possible system histories is denoted by $\mathcal{H}$. For any history h, the *Markov property* requires,

$$Pr(S^{t_{n+1}} \mid S^{t_n}, S^{t_{n-1}}, ..., S^{t_0}) = Pr(S^{t_{n+1}} \mid S^{t_n})$$

.

The system states enable a *factored representation* if the state space is characterized by a set of *variables* or *fluents* $\mathcal{F} = \{F_1, ..., F_l\}$. The value of each fluent F_i is denoted by f_i. In FSTS models, one way to identify these fluents is to utilize the internal structure of each task. Assume that a list of tasks from T_1 to T_m in $\mathcal{T}$ are required to be performed. Assume further that each task T_k contains at most q meta-methods. Divide the l fluents into m groups. $\mathcal{F}$ is consequently represented as an ordered list $(\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_m)$, where $\mathcal{F}_k$ for $\forall k \in \{1...m\}$ denotes a fluent group that contains a total of $q + 1$ fluents, and is used to represent the *working status* of task T_k. The first q fluents of $\mathcal{F}_k$ record the sequence of methods performed or being performed in order to achieve task T_k. The last fluent

indicates whether T_K has concluded. Since the future system state is determined only by the current working status of each task in $\mathcal{T}$, this factored representation of system states maintains the required *Markov property*.

Notice that this is only one potential representation of system states. There are possibly many other alternatives. Nevertheless, we do not care about the specific fluents adopted. The only issue that matters is that the factored representation of system states should maintain the *Markov property*. With the set of fluents $\mathcal{F}$, the *state space* $\mathcal{S}$ is defined as the set of all valid combinations of fluent values. Every such combination is represented by a list $(f_1, ..., f_l)$. It represents uniquely a state $S \in \mathcal{S}$.

3.4.2 Agents' Decisions

In FSTS, each agent's decision is expressed as a method sequence. However, under the DTP paradigm, and with the concept of system states, the decisions are more appropriate to be represented indirectly through decision policies. A *decision policy* (or *policy*) π is a mapping from the set of systems states to the decisions of performing certain domain method, that is, $\pi : \mathcal{S} \rightarrow \mathcal{M}$. One special method M^0 is included in $\mathcal{M}$ such that if π maps a state to M^0, then the agent applying the policy π will decide to perform no operations at that state.

An agent can only perform one domain method at a time. It makes a decision on selecting any new method only when it has no ongoing methods to perform. The decision process forms basically a loop. During each iteration, the agent updates its observation of the current system state, and uses the updated state information and policy π to identify the next domain method to perform. The agent executes the chosen method until the method has been finished. It then loops back to re-update its system state information and begins the next iteration.

3.4.3 The Stochastic Model of System Transitions

In DTP, the dynamics of a system is modeled by system transitions (or state changes). Given the global method sequence Seq or decision policy π and assume that the transitions are *stationary*, the system state changes can be abstracted with a transition matrix P. The matrix is of size $N \times N$, where N is the number of distinct system states. Every entry of the matrix P, p_{ij}, equals to $Pr(S^{t+1} = s_j \mid S^t = s_i)$, where s_i and s_j are two distinct system states. Using matrix P, the probability of obtaining any system history h under the policy π is evaluated through equation (3.2),

$$Pr(h \mid \pi) = \prod_{i=0}^{l} [(P)^i \cdot \vec{p}^0]_{S^{t_i}} \tag{3.2}$$

where $\vec{p}^0$ is the vector representing the initial distribution over the system states, and l is the history length. The operator $[\vec{p}]_S$ returns the probability of reaching state S, which is listed in the vector $\vec{p}$. With equation (3.2), it is obvious that the system transitions can be fully characterized by the vector $\vec{p}^0$ and the transition matrix P. $\vec{p}^0$ is given *a priori* by system designers. In the FSTS model, it refers to the probability for the agents to complete various groups of tasks $\mathcal{T}$. The transition matrix P is determined by the internal structure of each task T, that is, $(\{\overline{M}\}, \{g\})$. The set $\{\overline{M}\}$ regulates the potential domain methods that can be performed in order to complete task T. The system constraints imposed on task T, $\{g\}$, affect the execution of T through:

1. concluding the task T if certain hard system constraint is violated or all the required domain methods have been performed; or

2. changing the characteristics (e.g. processing time and success rate) of certain domain method to be performed in the future.

Since the fulfillment of each task $T \in \mathcal{T}$ is *independent*, the system constraints play an important role in determining the overall system transitions through determining the execution of each task.

Unlike a DTP problem, it may not be possible to derive accurately the transition matrix P from the FSTS model. The model of system constraints belongs to the subjective knowledge of each agent. The exact impact of these constraints might not be known *a priori*. This difference makes the traditional dynamic programming techniques inapplicable. One solution to this problem is to depend the agents' decision-makings upon the reinforcement learning, which is widely accepted as effective *model-free* alternatives for dynamic programming problems (see Chapter 4). Notice that the system constraint information is important for the learning agents. Compared with other fluents, such information is more significant in determining the rewards of performing any domain methods:

1. Since the violation of any hard system constraint will induce a task failure, a method that leads to such violations will be offered with a low reward (possibly a penalty) and is apparently unacceptable.

2. Soft constraints affect the way agents perform future methods. The relevance to certain soft constraints will become useful decision-making criteria for the agents when the methods under consideration are similar (e.g. they belong to the same meta-method of a task and violate no hard constraints).

3.4.4 The Quality of the Agents' Decisions

To quantify the *quality* of an agent's decisions, the *value function* $V(\bullet)$ is used in the DTP paradigm to map the set of system histories $\mathcal{H}$ into real number. The function is assumed to be *time-separable* [37]. For any particular system history h, $V(\bullet)$ is represented as a combination of all the rewards incurred when the system evolves according to h. We define the *reward function* $R : \mathcal{S} \times \mathcal{S} \rightarrow \Re$ to be a mapping that associates a *reward* with any system transitions. $R(S_1, S_2)$ returns 0 if the system transition from state S_1 to S_2 induces no task conclusion. For any history h of length l, the value of the history, $V(h)$, is defined as

$$V(h) = \sum_{i=0}^{l-2} \alpha(t_{i+1}) \cdot R(S^{t_i}, S^{t_{i+1}})$$

where $\alpha(t)$ is the *discount rate*. It gives a reward incurred in the far future a lower discount such that the agents place more focus on their short-term performance. For distinct decision policies, the length of obtainable system histories might not be identical. To tackle this difference, each history h is extended into an infinite-length list by repeatedly appending the last system state of h to itself. $V(h)$ is consequently re-defined as

$$V(h) = \sum_{i=0}^{\infty} \alpha(t_{i+1}) \cdot R(S^{t_i}, S^{t_{i+1}}) \tag{3.3}$$

With this definition of $V(h)$, the *expected value* of any policy π is evaluated according to:

$$EV(\pi) = \sum_{h \in \mathcal{H}} V(h) \cdot Pr(h \mid \pi) \tag{3.4}$$

where $Pr(h \mid \pi)$ is defined in equation (3.2). In the FSTS model, as each reward indicates the satisfaction of certain system goals, a better coordinated multiagent system is generally expected to achieve a higher total reward. Additionally, the degree of coordination can be further improved by adjusting the decision policies. With these characterizations, the formulation of expected values given in equation (3.4) can be directly applied by the FSTS model to measure the degree of coordination among agents. Taking this view, any agent coordination algorithm actually attempts to improve this measure.

3.5 Modeling Example: A multi-agent pathology lab system

To demonstrate the major modeling concepts in FSTS, a multi-agent pathology lab system is considered in this section. This system will also be used to examine

the effectiveness of the two reinforcement learning algorithms described in the next chapter. A pathology lab is a task-oriented environment. The objective of the lab is to produce pathological specimens, which are obtained from the original pathological samples through a series of inter-related domain operations. A typical pathology lab contains many devices designed for various domain operations. A simple conceptualized pathology lab is shown in Figure 3.2, and a simulated multi-agent system is developed to show the modeling techniques and experiment with the coordination methods.

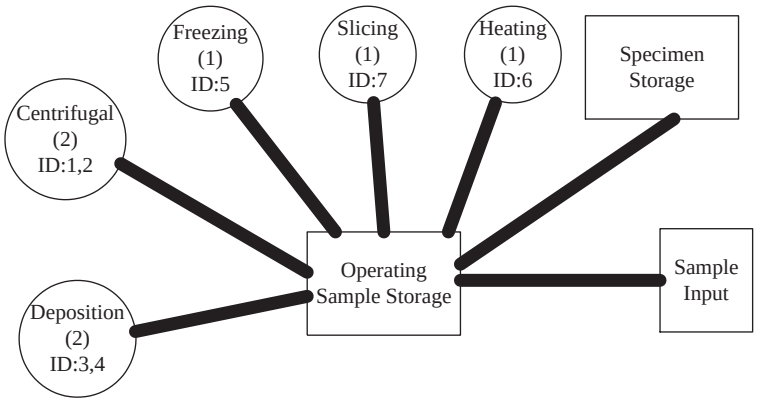

Figure 3.2: The structure of a simplified pathology lab.

In Figure 3.2, each circle represents a specific type of device designed for certain domain operations. It includes also the number of such devices and their IDs. The devices are arranged around the sample storage unit and connected to it through conveyor belts for transporting pathological samples. The samples are initially stored in the sample storage unit, and are later passed to the devices to carry out the necessary domain operations. After all the operations required by a given sample are completed appropriately, the produced specimen will be stored in the specimen storage. The output of the lab is taken from the storage. New samples are the input to the lab, and are supplied to the system through the sample input unit.

A multiagent system is designed to automate the operation of the pathology lab. For the simple case, two agents ($Agent1$ and $Agent2$) are deployed. $Agent1$ manages two centrifugal and one freezing devices. The rest of devices (Ref. Figure 3.2) are managed by $Agent2$. By design, each agent is only capable of performing one domain operation with a device at a time.

Following FSTS, each domain operation performed on a specific sample is an atomic operation and is modeled as a *method*. A *meta-method* $\overline{M}$ is described as a tuple of three components, $(\tilde{A}_c, \tilde{E}_q, \tilde{O})$, where $\tilde{E}_q$ and $\tilde{O}$ denote a fuzzy set of devices and a fuzzy set of pathological samples respectively. In our implementation of the pathology lab system, each $\tilde{O}$ contains only one pathologic sample. $\tilde{E}_q$ is

also a simple set of all the devices of the same type. $\tilde{A}$ denotes a fuzzy set of *actions*. An action refers to the specific operation performed by a *method*. Any method M with an action a_c, required device r, and pathological sample o is considered belonging to $\bar{M}$ if

$$Similar_{\overline{M}}(M) = \mu_{\tilde{A}_c}(a_c) \bigwedge \mu_{\tilde{E}_q}(r) \bigwedge \mu_{\tilde{O}}(o) > 0$$

where $\mu_{\tilde{A}_c}$, $\mu_{\tilde{E}_q}$, and $\mu_{\tilde{O}}$ are the *membership functions* of the fuzzy sets $\tilde{A}_c$, $\tilde{E}_q$, and $\tilde{O}$, and $\bigwedge$ is the standard *T-norm operator* [295]. The simulated system can process a total number of six types of samples, (from A to F), each having different sizes: (1) large sizes; (2) middle-sizes; and (3) small-sizes. As a convention, we use the symbol $A.1$ to denote large-sizes samples of type A.

To process these samples, the system is designed to have 25 distinct actions divided into 5 categories. Table 3.1 summarizes the actions for each categories, the necessary processing time, and the required devices. The processing time is measured in standard time intervals. The time interval is the basic unit of time that is sufficient for quantifying any domain-related time information in the system. To further illustrate the notion of meta-methods, Table 3.2 lists all the meta-methods of a task that produces a specimen from a sample o of type $A.1$. In Table 3.2, $\tilde{A}_c$ is the fuzzy set of actions. The nominator in the description of each component of $\tilde{A}_c$ denotes the membership degree of that component with respect to $\tilde{A}_c$.

Action Category	Centrifugal Operation	Deposition Operation	Freezing Operation	Heating Operation	Slicing Operation
Processing Time & Action ID	5–C_1,6–C_2, 7–C_3,8–C_4, 9–C_5	5–D_1,6–D_2, 7–D_3,8–D_4, 9–D_5	8–F_1,9–F_2, 10–F_3,11–F_4, 12–F_5	3–H_1,4–H_2, 5–H_3,6–H_4, 7–H_5	1–S_1,2–S_2, 3–S_3,4–S_4, 5–S_5
Device (ID)	1 or 2	3 or 4	5	6	7

Table 3.1: All applicable actions, their action category and processing time.

Meta-method	$\tilde{A}_c$	$\tilde{E}_q$	$\tilde{O}$
$\overline{M}_1$	$\{\frac{0.22}{C_1} + \frac{0.37}{C_2} + \frac{0.61}{C_3} + \frac{1}{C_4} + \frac{0.61}{C_5}\}$	{1,2}	$\{o\}$
$\overline{M}_2$	$\{\frac{0.22}{D_1} + \frac{0.37}{D_2} + \frac{0.61}{D_3} + \frac{1}{D_4} + \frac{0.61}{D_5}\}$	{3,4}	$\{o\}$
$\overline{M}_3$	$\{\frac{0.61}{S_1} + \frac{0.78}{S_2} + \frac{1}{S_3} + \frac{0.78}{S_4} + \frac{0.61}{S_5}\}$	{7}	$\{o\}$

Table 3.2: The set of meta-methods of task T for large sizes sample of type A.

The successful fulfillment of a task T depends on the satisfaction of certain stringent constraints imposed by the specimen production process. Some constraints are modeled in terms of *enable* relations (one kind of *hard relations*).

They constrain the order in which the methods in different meta-methods are performed. As an example, the hard relations among the meta-methods in Table 3.2 are modeled as a *directed graph*(Figure 3.3), which can be viewed as a work-flow. It stipulates that the methods in $\overline{M}_1$ should be performed first, while the methods in $\overline{M}_3$ can only be executed after the methods in $\overline{M}_2$ have completed.

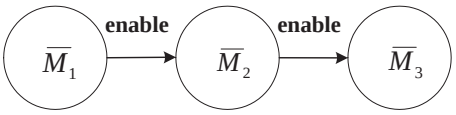

Figure 3.3: The *hard relations* among the meta-methods of task T. (Ref. Table 3.2)

In addition to the hard relations, one type of soft relation exists. Specifically, the execution of a centrifugal operation ($\overline{M}_4$), which consumes a long period of time, will improve the success rate and reduce the execution time of the deposition operation ($\overline{M}_5$) that follows. The soft relation is represented as a directed graph in Figure 3.4.

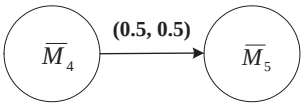

Figure 3.4: The *soft relation* between centrifugal and deposition meta-methods.

Equation (3.1) is used to measure the extent to which a method is relevant to certain hard relations. The relevance with respect to any soft relations is determined through a graph similarity measure to be discussed in Chapter 4. The evaluated degree is interpreted as the *possibility* that the method may be affected by the corresponding relations, and takes its value between 0 and 1. This *possibility* belongs to agents' subjective knowledge. The actual impact is to be estimated through the two reinforcement learning algorithms introduced in the next Chapter.

Each time the multi-agent pathology lab system starts, a given number of samples are supplied to the lab. Their types and sizes are randomly chosen. A distinct task is created for each sample. The overall objective of the system is to successfully fulfill all these tasks. For every task T that produces one specimen, equation (3.5) is used to evaluate the reward incurred when the task concludes.

$$Reward_T = \begin{cases} rFac - pFac & when\ task\ T\ is\ fulfilled \\ failPenalty & when\ task\ T\ fails \end{cases} \tag{3.5}$$

where *failPenalty* is the amount of punishment incurred after the task T fails, and *rFac* and *pFac* are the reward (positive) and penalty (negative) parts applicable to the fulfillment of T, and are evaluated through equation (3.6):

$$
\begin{aligned}
rFac &= Str \cdot \left(e^{\frac{t_b - t}{t_b} \cdot rDis} \right) \\
pFac &= Str \cdot \left(e^{\frac{t_b - t}{t_b} \cdot pDis} - e^{2 \cdot \frac{t - t_b}{t_b}} \right)
\end{aligned}
\tag{3.6}
$$

In (3.6), t denotes the actual time when task T concludes. $rDis$ and $rPis$ are two weight factors. Str is the *amplitude* of the rewards/penalties. t_b is the desirable conclusion time of task T, and is determined before the pathology system starts. Given the rewards of concluding each task, the overall performance of the system is measured according to equation (3.3). The discount rate $\alpha(t)$ is defined as $\alpha(t) = e^{-\alpha^c \cdot t}$, where α^c is a given discount constant.

With the above descriptions, it is evidenced that using the FSTS, the coordination problem in a multi-agent pathology lab system is explicitly presented from each agent's subjective perspective. Any coordination methods to be used in this situation is actually to improve a system performance measure defined in equation (3.3). Due to the generality of the pathology lab application (e.g. many manufacturing systems can be described similarly), it is also evidenced that the FSTS model is flexible in dealing with the uncertainties and interdependencies between agents and their environments.

3.6 Summary

This chapter presented a Fuzzy Subjective Task Structure (FSTS) model to help agents identify coordination problems in a task-oriented domain through their subjective beliefs and intentions with uncertainties. The model is first introduced in concept and then applied to a multi-agent pathology lab system to show its usefulness. With FSTS, the agent coordination is a Decision Theoretical Planning (DTP) problem, where reinforcement learning methods are ideal to be used.

The proposed FSTS model is effective in dealing with the uncertainties agents have about themselves and the environment. It is developed for task-oriented domains and is widely applicable. Due to the extreme complexities of real-life applications, the model is expected to be extended to include other features essential to agent coordination. For example, in a multi-agent environment, the subjective beliefs and intentions of an agent are constantly changing. Sometimes, uncertainties about one thing may lead the agent to collect more information about it, and thus to be more certain of accessing its impacts (modeling information retrieval behavior). In addtion, agents may exchange their information in order to be more clear about the current situation (modeling agent communication). They may also try to learn the outcomes of their decisions so as to enhance their performance when similar situations occur (modeling learning and decision making). Without

changing the FSTS model, The last two issues are covered respectively in Chapter 4 and 5 with proper coordination methods.

Chapter 4

Agent Coordination via Reinforcement Learning

4.1 Introduction

This chapter utilizes reinforcement learning methods to address coordination issues in a *cooperative* task-oriented environment [90], where multiple agents cooperate to fulfill the system goals by performing a series of tasks. The existence of constraints in such a environment affects the potential task fulfillment behaviors of each agent. As described in Chapter 3, the constraints can be classified into *hard constraints* and *soft constraints*. The hard constraints are stringent. Their violation will induce the failure of the corresponding tasks. The soft constraints are soft in a sense that their violation may affect task fulfillment, but not necessarily results in a failure. The interdependencies among agents are derived from the constraints imposed on the on-going tasks.

Two learning algorithms, "*coarse-grained*" and "*fine-grained*", are proposed to address the *hard* and *soft* constraints, respectively [59, 62]. The "*coarse-grained*" algorithm is designed to identify the agents' decision-making policies that respect only hard system constraints. The "*fine-grained*" learning algorithm is applied after the hard system constraints have been dealt with. It exploits the learning results from the coarse-grained algorithm, and takes explicitly the soft constraints information into consideration when adjusting the corresponding policies. In effect, the two algorithms constitute two separate steps of the whole learning process to address both hard and soft constraints. As the experiments demonstrate, this effectively improves the decision quality and performance of agents.

The approach presented in this Chapter contributes to the previous learning-based coordination techniques (e.g., in [25, 277]) in the following two aspects. First, it is designed for general task-oriented environments in which the system

• Part of the research results shown in this chapter appears in the international journal of *Autonomous Agents and Multi-Agent Systems*, 2005.

constraints dynamically determine the interdependencies among agents [100]. The constraints are explicitly modeled in FSTS (see Chapter 3) and handled by the learning algorithms. Second, the whole coordination issue is dealt with through the combination of two learning algorithms (coarse-grained and fine-grained), while only one learning algorithm is typically explored in the literature. In essence, our two-steps learning approach to the total learning work parallels with the system abstraction method reported in [88]. Particularly, the various system constraints can be characterized as a group of *variables* or *fluents*. A reward function is provided to measure the system performance in terms of system state changes. As the violation of hard constraints has a fatal consequence to the overall system performance, the fluents characterizing hard constraints are more closely related with the reward function. Agents are therefore trained initially in a simulated environment that emphasizes only hard constraints. This is a separation of concern, as different constraints impact the satisfaction of system goals at different levels and require different ways to be addressed.

Because of the *curse of dimensionality* [23], it is necessary to adopt certain approximation methods for practical learning applications [236]. In this chapter, the "*abstract system states*", a new set of system states, are obtained from the concrete system states by *state aggregation*. State aggregation can be considered as one of the approximation techniques. In many domains, it has been shown to reduce exponentially the original problem size and make the traditional dynamic programming and reinforcement learning approaches directly applicable [36]. When designing the two learning algorithms, four new assumptions are made, which serve as the aggregation criteria. They guide system designers to identify similar states and aggregate them together. In the deterministic environment, the first three assumptions are analogous to but less stringent than the notion of equivalence identified in [125], while the fourth assumption can often be omitted practically. In this Chapter, a bound on the difference between the learning result obtained from the abstract system and the one from the concrete system is presented. It is interesting to note that these assumptions can serve as a proper aggregation concept and breaking them may introduce more deviations between the two learning results.

This Chapter is organized as follows. Section 4.2 studies the assumptions on which the two learning algorithms are based. A high-level overview of the algorithms are given in Section 4.3. The explicit explanation of each algorithm is arranged in Section 4.4 and Section 4.5, respectively. Section 4.6 examines theoretically the convergence properties of the learning algorithms. Experiment evaluation of their effectiveness is covered in Section 4.7. A discussion on the related work is presented in Section 4.8. Finally, Section 4.9 concludes this Chapter.

4.2 Learning Objectives and Basic Assumptions

As described in Chapter 3, the agents' decision-making policy π is a function that maps each system state to a specific domain method. Rather than directly constructing such a policy, one alternative that conventional reinforcement learning algorithms take is to estimate the expected value of the policy π at any system states (or state changes). Particularly, the expected value of the policy π at state S is defined as

$$V^{\pi}(S) = \sum_{S' \in \mathcal{S}} \{Pr(S' \mid \pi(S), S) \cdot \alpha_{S'} \cdot [R(S, S') + V^{\pi}(S')]\} \tag{4.1}$$

where $\alpha_{S'}$ denotes the discount rate at the time point when state S' is reached from S. $Pr(S' \mid \pi(S), S)$ is the probability of reaching system state S' if the agent makes a decision based on the policy π at state S. From equation (4.1), the method to perform on state S can be indirectly determined through equation (4.2):

$$\pi(S) = \underset{M \in \mathcal{M}}{argmax} \ (\textstyle\sum_{S' \in \mathcal{S}} \{Pr(S' \mid M, S) \cdot \alpha_{S'} \cdot [R(S, S') + V(S')]\}) \tag{4.2}$$

where $V(S)$ is the expected value of state S. Apparently, for any mapping $V : \mathcal{S} \rightarrow \Re$, there exists a corresponding policy π that satisfy equation (4.2). Constructing the function $V(S)$ for every $S \in \mathcal{S}$, agents indirectly identify their decision-making policies. Besides the function $V(\bullet)$, another alternative for reinforcement learning algorithms is to identify the so-called *quality function* Q. Since the reward function is defined over all potential state changes, the quality function is presented in the form $Q(S, S', M)$, which estimates the expected value of performing method M that transforms the system state from S to S'. Given the quality function $Q(S, S', M)$, the method to perform at state S should satisfy the following equation (4.3):

$$\pi(S) = \underset{M \in \mathcal{M}}{argmax} \ \{\textstyle\sum_{S' \in \mathcal{S}} [Pr(S' \mid M, S) \cdot Q(S, S', M)]\} \tag{4.3}$$

Equation (4.3) evaluates the expected value of performing each method M at system state S. The method that achieves the highest value will be chosen to be performed. Using a state pair in the quality function eliminates the necessity of enumerating the quality of performing every method. Suppose, for example, that within a given pair of states only the status of two tasks have been changed. It is unnecessary to evaluate the qualities of performing methods that do not contribute to these two tasks. In the simplest cases, the actual method to be evaluated by

$Q(S, S', M)$ may not need to be explicitly distinguished. What matters is the state changes.

Equation (4.3) indicates that its use requires agents to identify *a priori* the stochastic model of system transitions. To address this issue, the following equation (4.4) is used to decide the method to be performed at each system state.

$$\pi(S) = \underset{M \in \mathcal{M}}{argmax} \left\{ \underset{S' \in \mathcal{S'}_{S,M}}{Avg} [Q(S, S', M)] \right\} \tag{4.4}$$

where $\mathcal{S'}_{S,M}$ is a specific set of system states, $\{S' \mid Pr(S' \mid M, S) > 0\}$. Following equation (4.4), agents choose the method that gives the highest *average* quality value. This decision strategy requires no *prior* stochastic transition knowledge. Similar strategies (to determine a method without following equation (4.2) or (4.3)) were also explored in the literature, such as the risk-sensitive reinforcement learning algorithm [134]. Taking into account equation (4.4), the learning objectives of our two learning algorithms are to identify a quality function $Q^*(S, S', M)$ that satisfies equation (4.5):

$$Q^*(S_1, S_2, M) = R(S_1, S_2) + \alpha_{S_2} \cdot \underset{(S_3, M') \in \mathcal{S}_{S_2}}{Avg} (Q^*(S_2, S_3, M')) \tag{4.5}$$

where $\mathcal{S}_{S_2}$ refers to a set of state-method pairs, such that for every pair (S_3, M') in the set, $Pr(S_3 \mid M', S_2) > 0$. The methods are characterized in terms of a group of *fluents*, called method fluents. Typical fluents may indicate, for example, the particular meta-method to which the method belongs and the potentials of satisfying certain system constraints if the method is performed. In this manner, the triplet (S, S', M) for a quality function Q is identified by three respective groups of fluents. The space size of the triplets equals to the number of valid combinations of the fluent values. In practical applications, this space size can be quite large. To handle this issue, one approach is to aggregate similar system states and domain methods so as to form an *abstract state space* and an *abstract method space*, respectively. This aggregation approach is in fact a kind of partitions on both sets $\mathcal{S}$ and $\mathcal{M}$.

In general, a partition E of the set $\mathcal{S} = \{S_1, ..., S_n\}$ is a set of sets $\{\mathcal{B}_1, ..., \mathcal{B}_r\}$ such that each $\mathcal{B}_i$ is a subset of $\mathcal{S}$, $\mathcal{B}_i$ are disjoint from one another, and the union of all $\mathcal{B}_i$ returns the set $\mathcal{S}$. Each member of a partition is called a *block*. In the succeeding discussion, each block of system states is denoted as $\mathcal{B}^S$ and each block of domain methods is denoted as $\mathcal{B}^M$. Given the partitions, our learning algorithms to be described later work for the *aggregated quality function* of the form $Q^*(B^S{}_1, B^S{}_2, B^M)$, instead of the quality function $Q^*(S, S', M)$, as follows:

$$Q^*(B^S{}_1, B^S{}_2, B^M) = \underset{S_1 \in B^S{}_1, S_2 \in B^S{}_2, M \in B^M, Pr(S_2|M,S_1)>0}{Avg} (Q^*(S_1, S_2, M)) \quad (4.6)$$

In equation (4.6), the quality of the triplet $(B^S{}_1, B^S{}_2, B^M)$ is defined as the *average* of the qualities of all the triplets (S_1, S_2, M) such that $S_1 \in B^S{}_1$, $S_2 \in B^S{}_2$, $M \in B^M$, and $Pr(S_2 \mid M, S_1) > 0$. This definition facilitates a direct extension from the learning algorithm based on function $Q^*(S, S', M)$ to one based on function $Q^*(B^S{}_1, B^S{}_2, B^M)$. Notice that the use of the average operation in estimating the expected value at each system state is also recommended by some DTP researchers [36]. With equation (4.6), the method selected at each system state S is to satisfy equation (4.7) as follows:

$$\pi(S) = \underset{M \in \mathcal{M}}{argmax} \left\{ \underset{\substack{for\ all\ S' \in \mathcal{S}'_{S,M} \\ S \in B^S, S' \in B^{S'}, M \in B^M}}{Avg} [Q^*(B^S, B^{S'}, B^M)] \right\} \quad (4.7)$$

Equation (4.7) follows equation (4.4) simply by replacing each quality $Q(S, S', M)$ with its corresponding aggregated quality function. It should be noted that both equations impose nearly no requirements for agents to know *a priori* the stochastic model of system transitions. However, in order to utilize them in practice, the agents may still need to answer the following question: given a system state S and a domain method M to perform in S, which system state S' can probably be reached when M is finished? Facing this question, one alternative is to avoid it by making a few assumptions regarding the partition of sets $\mathcal{S}$ and $\mathcal{M}$. In practice, the partition is often represented by a group of *partition fluents*, i.e., the system states or domain methods are classified into different blocks if they induce differed partition fluent values. The partition fluents may be selected directly from the fluents characterizing system states or domain methods. They may also work as functions that take as their input variables multiple state fluents (or method fluents). Selecting varied partition fluents leads to different system partitions. To ease the discussion, assume that:

A^* If when method M starts to perform in system state S, the expected system state when M finishes is S' (denoted by $S \xrightarrow{M} S'$), then S' is said to be reachable from S with probability 1 at the time when M finishes.

The assumption A^* can be relaxed by requiring that for any possible state-method pair (S, M), there exists one state block B^S such that $\sum_{S' \in B^S} Pr(S' \mid M, S) = 1$. Nevertheless, the discussion of following partition assumptions is based on the assumption A^*, and the two learning algorithms are designed to

handle the violation of A^*. The four assumptions regarding the system partitions are presented as follows:

A1 If $S \xrightarrow{M} S'$, $S \in B^S$, $S' \in B^{S'}$, and $M \in B^M$, then $B^S \xrightarrow{B^M} B^{S'}$.

A2 If $B^S \xrightarrow{B^M} B^{S'}$, then for any $S \in B^S$, there exist $S' \in B^{S'}$ and $M \in B^M$, such that $S \xrightarrow{M} S'$.

A3 If $S \xrightarrow{M} S'$, $S \in B^S$, $S' \in B^{S'}$, and $M \in B^M$, then there exist no other M', $M' \in B^M$, such that $S \xrightarrow{M'} S''$, $S'' \in B^{S'}$.

A4 if $B^S \xrightarrow{B^M} B^{S'}$, then among all the triplets (S, S', M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$, each $S' \in B^{S'}$ appears exactly once.

Note that $B^S \xrightarrow{B^M} B^{S'}$ implies generally the existence of a triplet (S, S', M) such that $S \xrightarrow{M} S'$, $S \in B^S$, $S' \in B^{S'}$, and $M \in B^M$. Suppose that for two system blocks B^S and $B^{S'}$, and one method block B^M, $B^S \xrightarrow{B^M} B^{S'}$ is valid, the assumptions A1–A3 essentially require that for each state $S \in B^S$, there must exist another state $S' \in B^{S'}$ such that by performing a method in B^M, S' is reachable from S. Meanwhile, for any other methods in B^M, they are either inapplicable in state S or the expected system state after method execution lies outside of $B^{S'}$ (Ref. Figure 4.1).

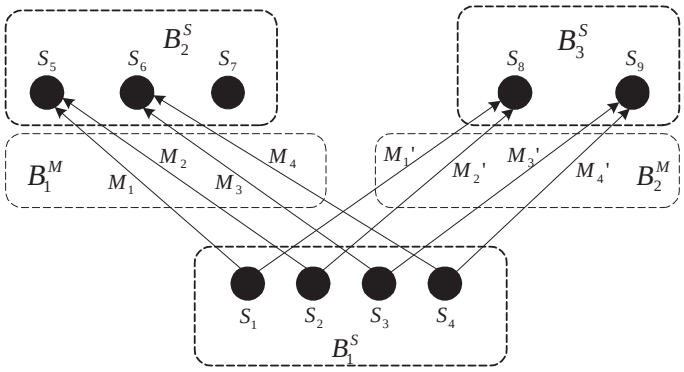

Figure 4.1: An example partition of the system states and methods that satisfies assumptions A1–A3.

Figure 4.1 shows a partition of three state blocks and two domain method blocks. There are two groups of system transitions, denoted as ${B^S}_1 \xrightarrow{{B^M}_1} {B^S}_2$ and ${B^S}_1 \xrightarrow{{B^M}_2} {B^S}_3$ respectively. In order to satisfy assumption A1, for each system state S in ${B^S}_1$, there exists at least one state S' in ${B^S}_1$ or ${B^S}_2$ such that after performing certain method in ${B^M}_1$ or ${B^M}_2$, S' is reachable. The satisfaction of assumption A3 further regulates that if S' is reachable from S, there exists no other method in ${B^M}_1$ or ${B^M}_2$ such that by performing that method, S' is still reachable from S. That is to say that given any system state S, the

effects of performing the various domain methods in S (i.e., the system state after execution) have been classified into different state blocks. This literally avoids the case that prevents the agents from using equation (4.7). A triplet of blocks $(B^S, B^{S'}, B^M)$ is said applicable if $B^S \overset{B^M}{\rightarrow} B^{S'}$. Assume without loss of generality that the aggregated quality function is defined over all applicable triplets, $(B^S, B^{S'}, B^M)$. The *average* quality of performing one method $M \in B^M$ at system state $S \in B^S$ can be expressed as

$$\underset{\substack{for\ all\ B^{S'} \\ S \in B^S, M \in B^M}}{Avg} [Q^*(B^S, B^{S'}, B^M)] \tag{4.8}$$

From equation (4.8), the qualities of all the applicable triplets $(B^S, B^{S'}, B^M)$ such that $S \in B^S$, $M \in B^M$ are averaged to provide an average quality of performing a method $M \in B^M$. Compared with equation (4.7), this equation requires no information about system transitions. The method that achieves the highest value in the equation is the one to be selected. Notice that the three assumptions (A1–A3) are similar to the *stochastic bisimulation* concept ([125]). More specifically, the stochastic bisimulation for aggregating system states requires:

B1 for each state $S \in B^S$, $R(S)$ is well-defined and equals to $R(B^S)$.

B2 for any two states S_1 and S_2 in the same block B^S, and for any block $B^{S'}$ and method M, $Pr(B^{S'} \mid M, S_1) = Pr(B^{S'} \mid M, S_2)$.

where $Pr(B^{S'} \mid M, S) = \sum_{S' \in B^{S'}} Pr(S' \mid M, S)$. If each method block B^M is considered as an individual domain method, assumptions A2 and A3 are actually identical with B2 above. In fact, assumption A3 extends B2 by describing the *bisimulation* relation in terms of method blocks rather than individual methods. Despite the apparent similarity, our approach differs from the bisimulation relation in that the reward function is defined over all potential state changes. In addition, the reward, $R(B^S, B^{S'})$, over any state pair (S_1, S_2) with $S_1 \in B^S$, $S_2 \in B^{S'}$ is not required to be a fixed value. In this sense, our assumptions A1–A3 can be considered as a relaxed version of the stochastic bisimulation. However, the assumption A4 is much more stringent (stronger) than the stochastic bisimulation condition. It imposes a one-to-one correspondence among the system states of different state blocks as shown in Figure 4.2. Nevertheless, the violation of A4 does not affect the applicability of equation (4.8).

The major difference between Figure 4.2 and 4.1 lies in that for any system state S' that belongs to block ${B^S}_2$ or ${B^S}_3$, there exists a unique state S of block ${B^S}_1$ such that S' is reachable from S by performing a method in ${B^M}_1$ or ${B^M}_2$. The significance of this one-to-one correspondence is the ability to reduce the

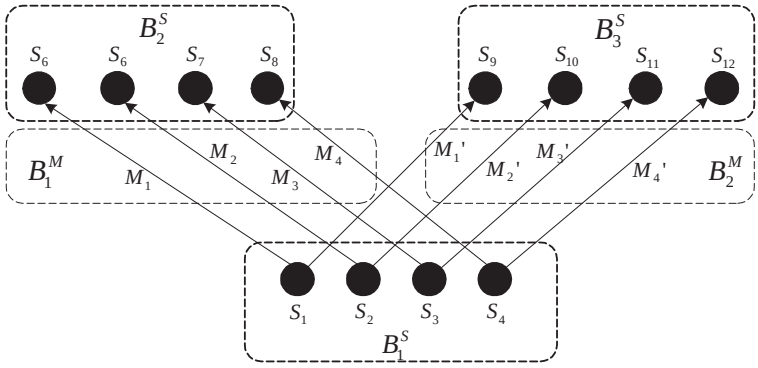

Figure 4.2: An example partition of the system states and methods that satisfies assumptions A1–A4.

variations between the two quality functions $Q^*(S, S', M)$ and $Q^*(B^S, B^{S'}, B^M)$. To see this, notice that for any system partition E that satisfies assumptions A1–A3, there exists a refined partition E' of E such that assumption A4 is also valid. A refined partition tends to more accurately estimate the original system. Two error bounds are provided later for these assumptions. The first bound refrains the maximum error between $Q^*(S, S', M)$ and $Q^*(B^S, B^{S'}, B^M)$, and the second bound indicates that an extra gap might be introduced into the error when the partition violates assumption A4. Similar error bounds were given in [88]. These error bounds validate the importance of assumption A4, which also helps when describing the two learning algorithms.

Suppose there exists at least one method block B^M, which links two state blocks B^S and $B^{S'}$ by making $B^S \xrightarrow{B^M} B^{S'}$ valid. The *reward span* for the two blocks B^S and $B^{S'}$ is defined as the maximum range of possible rewards:

$$span(B^S, B^{S'}) = \max_{S_1 \in B^S, S_2 \in B^{S'}} [R(S_1, S_2)] - \min_{S_1 \in B^S, S_2 \in B^{S'}} [R(S_1, S_2)]$$

The maximum reward span for a partition E is then

$$\delta = \max_{(B^S, B^{S'})} \left(span(B^S, B^{S'}) \right)$$

Theorem 1 *For any triplet (S_1, S_2, M) such that $S_1 \xrightarrow{M} S_2$, assume (1) the partition E satisfies assumptions A1–A4; (2) the difference between the quality functions $Q^*(S_1, S_2, M)$ and $Q^*({B^S}_1, {B^S}_2, B^M)$ is finite, then*

$$\left| Q^*(S_1, S_2, M) - Q^*({B^S}_1, {B^S}_2, B^M) \right| \leq \frac{\delta}{1 - \alpha}$$

where $S_1 \in {B^S}_1$, $S_2 \in {B^S}_2$, $M \in B^M$, and α is the maximum among all the discount rates α_S.

Proof. From traditional dynamic programming theories, the quality function $Q^*(S, S', M)$ defined in equation (4.5) is estimated through a sequence of quality functions Q^k, that is

$$Q^0(S, S', M) = R(S, S')$$
$$Q^k(S, S', M) = R(S, S') + \alpha_S \cdot \underset{(S'', M') \in \mathcal{S}_{S'}}{Avg} \left(Q^k(S', S'', M') \right)$$

The major step is to treat each average operation as an expectation operation, and the details are omitted here. Accordingly, the quality function $Q^*(B^S, B^{S'}, B^M)$ is estimated through another sequence of aggregated quality functions. That is

$$Q^k(B^S, B^{S'}, B^M) = \underset{S \in B^S, S' \in B^{S'}, M \in B^M, Pr(S'|M,S)>0}{Avg} \left(Q^k(S, S', M) \right)$$

This theorem is proved inductively by showing that for all k,

$$\left| Q^k(S, S', M) - Q^K(B^S, B^{S'}, B^M) \right| \leq \sum_{i=0}^{k} \delta \cdot \alpha^i$$

Since $Q^*(S, S', M) = lim_{k \to \infty} Q^k(S, S', M)$ and $Q^*(B^S, B^{S'}, B^M) = lim_{k \to \infty} Q^k(B^S, B^{S'}, B^M)$, the theorem is thus proved.

Since,

$$\left| Q^0(S, S', M) - Q^0(B^S, B^{S'}, B^M) \right| \leq \delta$$

the base of the induction is immediate. Now assume that for some fixed k,

$$\left| Q^k(S, S', M) - Q^k(B^S, B^{S'}, B^M) \right| \leq \sum_{i=0}^{k} \delta \cdot \alpha^i$$

therefore

$$\begin{aligned} & \left| Q^{k+1}(S, S', M) - Q^{k+1}(B^S, B^{S'}, B^M) \right| \\ = & \left| \left[R(S, S', M) + \alpha_{S'} \cdot \underset{(S'', M')}{Avg} \left(Q^k(S', S'', M') \right) \right] \right. \\ & \left. \left[\underset{(S, S', M)}{Avg} \left(Q^k(S, S', M) \right) \right] \right| \end{aligned} \tag{4.9}$$

In equation (4.9), the average operation is taken over all $Q^k(S, S', M)$ such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$ can be written as

$$\begin{aligned} & \underset{(S, S', M)}{Avg} \left(Q^k(S, S', M) \right) \\ = & \underset{(S, S', M)}{Avg} R(S, S') + \underset{(S, S', M)}{Avg} \alpha_{S'} \cdot \left[\underset{(S'', M')}{Avg} \left(Q^k(S', S'', M') \right) \right] \end{aligned}$$

where the first term of the above equation is identical with $Q^0(B^S, B^{S'}, B^M)$, and the last term of the equation can be expressed alternatively with the sequence Q^k of aggregated quality functions as

$$\begin{aligned} & \underset{S' \in B^{S'}}{Avg}\ \alpha_{S'} \cdot \left[\underset{(S'',M')}{Avg}\ \left(Q^k(S', S'', M') \right) \right] \\ & = \alpha_{S'} \cdot \underset{(B^{S''},B^{M'})}{Avg}\ \left(Q^k(B^{S'}, B^{S''}, B^{M'}) \right) \end{aligned} \tag{4.10}$$

With these transformations, equation (4.9) is extended as

$$\begin{aligned} & \left| Q^{k+1}(S, S', M) - Q^{k+1}(B^S, B^{S'}, B^M) \right| \\ \leq\ & \left| Q^0(S, S', M) - Q^0(B^S, B^{S'}, B^M) \right| + \\ & \alpha_{S'} \cdot \left| \underset{(S'',M')}{Avg}\ (Q^k(S', S'', M')) - \underset{(B^{S''},B^{M'})}{Avg}\ (Q^k(B^{S'}, B^{S''}, B^{M'})) \right| \\ \leq\ & \delta + \alpha \cdot \underset{\substack{(B^{S''},B^{M'}) \\ S'' \in B^{S''}, M' \in B^{M'}}}{Avg} \left| Q^k(S', S'', M') - Q^k(B^{S'}, B^{S''}, B^{M'}) \right| \\ \leq\ & \delta + \alpha \cdot \sum_{i=0}^{k} \delta \cdot \alpha^i \\ =\ & \sum_{i=0}^{k+1} \delta \alpha^i \end{aligned} \tag{4.11}$$

The derivation of the above inequalities is from the fact that $Q^0(S, S', M) = R(S, S')$ and is based on the assumptions A1–A4 for partition E, particularly the one-to-one correspondence among system states of different state blocks. By the inductive hypothesis and inequality (4.11), the theorem is thus proved. **Q.E.D**

The violation of assumption A4 breaks the one-to-one correspondence among system states of different state blocks. Within all the triplets (S, S', M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$, any state $S' \in B^{S'}$ might appear from 0 to multiple times. As shown in Figure 4.1, state $S_7 \in B^S{}_2$ is not reachable from any states of $B^S{}_1$ by performing methods belong to $B^M{}_1$. On the other hand, states $S_5, S_6 \in B^S{}_2$ are reachable from two distinct states of $B^S{}_1$. This variation prevents the transformation given in equation (4.10). In order to maintain a bounded difference between the quality functions $Q(S, S', M)$ and $Q(B^S, B^{S'}, B^M)$, it is necessary to assume that certain information about the quality functions is available. Specifically,

$$max \left\{ \left| \begin{array}{c} \underset{(S,S',M)}{Avg} \left[\underset{(S'',M')}{Avg} \left(Q(S',S'',M')\right) \right] \\ - \underset{S' \in B^{S'}}{Avg} \left[\underset{(S'',M')}{Avg} \left(Q(S',S'',M')\right) \right] \end{array} \right| \right\} \leq \Delta \tag{4.12}$$

Equation (4.12) states that the bound between the two average operations is Δ. One operation is over the set of triplets (S, S', M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$. The other is over the system states belonging to the state block $B^{S'}$. With this restriction, the error bound for the case where the partition violates assumption A4 is decidable.

Theorem 2 *For any triplet (S_1, S_2, M) such that $S_1 \xrightarrow{M} S_2$, and assume (1) the partition E satisfies assumptions A1–A3; (2) equation (4.12) is valid with finite Δ, then*

$$\left| Q^*(S_1, S_2, M) - Q^*(B^S{}_1, B^S{}_2, B^M) \right| \leq \frac{\delta + \alpha \cdot \Delta}{1 - \alpha}$$

where $S_1 \in B^S{}_1$, $S_2 \in B^S{}_2$, and $M \in B^M$. α refers to a positive constant that is smaller than 1.

Proof. This result is proved inductively similar to the proof of Theorem 1. Therefore, the focus is put only on the inductive step. The inductive hypothesis is

$$\left| Q^k(S, S', M) - Q^k(B^S, B^{S'}, B^M) \right| \leq \sum_{i=0}^{k} \delta \cdot \alpha^i + \sum_{i=1}^{k} \Delta \cdot \alpha^i$$

which clearly holds for $k = 0$. The inductive step is treated as follows:

$$\begin{aligned} & \left| Q^{k+1}(S, S', M) - Q^{k+1}(B^S, B^{S'}, B^M) \right| \\ = & \left| \left[R(S, S') + \alpha_{S'} \cdot \underset{(S'',M')}{Avg} \left(Q^k(S', S'', M')\right) \right] \right. \\ & \left. - \left[\underset{(S,S',M)}{Avg} \left(R(S, S')\right) + \underset{(S,S',M)}{Avg} \left(\alpha_{S'} \cdot \underset{(S'',M')}{Avg} Q^k(S', S'', M') \right) \right] \right| \\ \leq & \left| Q^k(S, S', M) - Q^k(B^S, B^{S'}, B^M) \right| \\ & + \alpha \cdot \left| \underset{(S'',M')}{Avg} \left(Q^k(S', S'', M')\right) - \underset{(S,S',M)}{Avg} \left(\underset{(S'',M')}{Avg} Q^k(S', S'', M') \right) \right| \end{aligned}$$

Further expanding the last term of the above inequality,

$$\alpha \cdot \left| \underset{(S'',M')}{Avg} \left(Q^k(S', S'', M')\right) - \underset{(S,S',M)}{Avg} \left(\underset{(S'',M')}{Avg} Q^k(S', S'', M') \right) \right|$$

$$
\begin{aligned}
&\leq \alpha \cdot \left| \underset{(S'',M')}{Avg} \left(Q^k(S', S'', M')\right) - \underset{S' \in B^{S'}}{Avg} \left(\underset{(S'',M')}{Avg} Q^k(S', S'', M') \right) \right| + \alpha \cdot \Delta \\
&= \alpha \cdot \left| \underset{(B^{S''},B^{M'})}{Avg} \left(Q^k(B^{S'}, B^{S''}, B^M) - Q^k(S', S'', M') \right) \right| + \alpha \cdot \Delta \\
&\leq \sum_{i=1}^{k+1} \delta \cdot \alpha^i + \sum_{i=1}^{k+1} \Delta \alpha^i
\end{aligned}
$$

With the above inequality, the inductive step is concluded by showing that

$$
\begin{aligned}
&\left| Q^{k+1}(S, S', M) - Q^{k+1}(B^S, B^{S'}, B^M) \right| \\
\leq\ & \delta + \sum_{i=1}^{k+1} \delta \cdot \alpha^i + \sum_{i=1}^{k+1} \Delta \alpha^i \\
\leq\ & \sum_{i=0}^{k+1} (\delta + \alpha \cdot \Delta) \cdot \alpha^i
\end{aligned}
$$

By taking the above inequality in the limit that $k \to \infty$, the result of the theorem yields. **Q.E.D**

Compared with Theorem 1, Theorem 2 indicates that the violation of assumption A4 may introduce another gap into the difference between the quality function $Q^*(S, S'.M)$ and the quality function $Q^*(B^S, B^{S'}, B^M)$. Nevertheless, these error bounds are derived from worst cases. In actual applications, the difference introduced by a system partition E may be relatively smaller. In other words, assumption A4 might be omitted in practice, and only assumptions A1–A3 need to be ensured. The satisfaction of A1–A3 does not rely on the exact model of system transitions. The system designers can explore their domain knowledge to construct a partition that respect the three assumptions. Furthermore, as shown in Section 4.6, the violation of A1–A4 does not affect the convergence property of the coarse-grained learning algorithm. In this sense, they serve simply as a group of criteria that determines the properness of the partition fluents selected by system designers. The experiments described in Section 4.7 also demonstrate these ideas and show that the two learning algorithms work well without respecting assumption A4.

4.3 Overview of Learning Algorithms

Two learning algorithms ("coarse-grained" and "fine-grained" learning algorithms) maintain the final learning results in the form of Takagi-Sugeno fuzzy systems [259]. The coarse-grained learning algorithm is devised for managing *hard constraints*. Its general structure is illustrated in Figure 4.3.

With this structure (Ref. Fig. 4.3), the coarse-grained learning algorithm can insert, modify, or delete data items from a table of the database, which is used

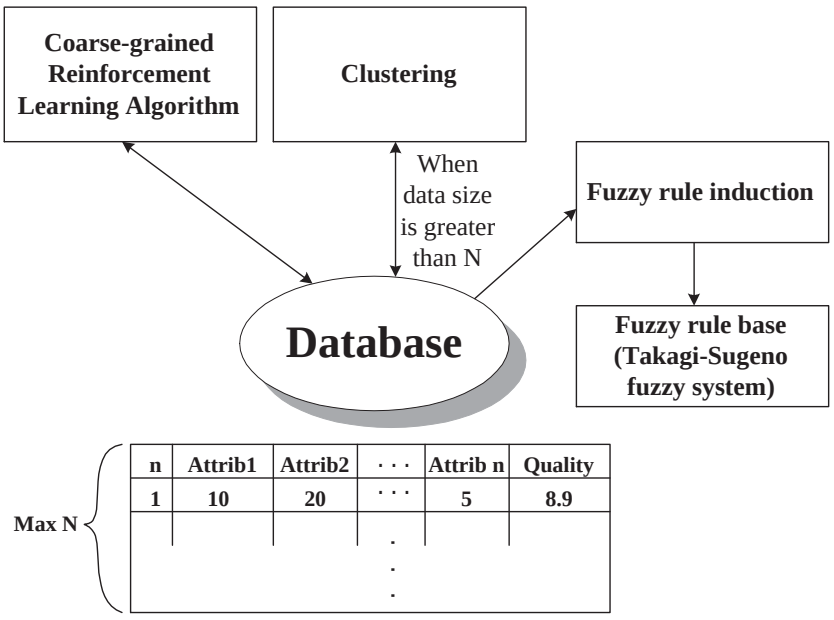

Figure 4.3: The general structure of the first learning algorithm.

for storing the temporary learning results. Each row of the table contains several fields for partition fluents and one field for the estimated *quality*. When some tasks are concluded, agents re-approximate the qualities of performed methods and store the derived information into the table. The size of the table is fixed to N. The *clustering method* is applied to combine similar items into a single one if the number of data items exceeds N. The final fuzzy rules are induced from the table by exploiting a fuzzy rule induction algorithm. It will lead to some information lost and therefore degrade the overall performance of the learning algorithm. However, if N is of a manageable size, each item of the table is applied to construct a distinct fuzzy rule in order to minimize the impact of the rule induction procedure. The algorithm is termed *coarse-grained* because the learning process is carried out assuming the existence of only hard constraints. Besides, the learning result is obtained from the simulated data before the actual running of the system.

The fine-grained learning algorithm exploits the linear architecture of the Takagi-Sugeno fuzzy system. It is designed for managing *soft constraints*, and is used to estimate the expected *quality changes* $\Delta Q'$ resulting from *soft constraints*. Since it considers both hard and soft system constraints, it is therefore termed as "*fine-grained*". The fuzzy rule base is constructed before the learning process begins. Whenever a method M is completed, agents re-estimate the quality of M and modify the corresponding fuzzy rules. This algorithm directly use the learning results from the *coarse-grained learning algorithm* and is adopted from the $TD(\lambda)$ algorithm with linear approximation architectures. The amount

of computation required during each learning step is adjustable (by incorporating a proper number of fuzzy rules) and can be comparably smaller than that for the *coarse-grained learning algorithm*.

By executing these two learning algorithms, the agents obtain the final learning result, $Q' + \Delta Q'$, which is the estimated quality when performing each method in the presence of both the hard and soft constraints. The overall relations between the two algorithms is illustrated in Figure 4.4. Upon determining which method to perform, agents first consult their coarse-grained learning results, then utilize their fine-grained learning algorithm to decide which method to perform. There are several ways to construct the decision policies from the learning results. In this paper, a simple approach is used: (1) during the learning process, the method most probably selected to perform is the one with the highest average quality; and (2) during the system execution, the method with the highest average quality is selected with probability 1.

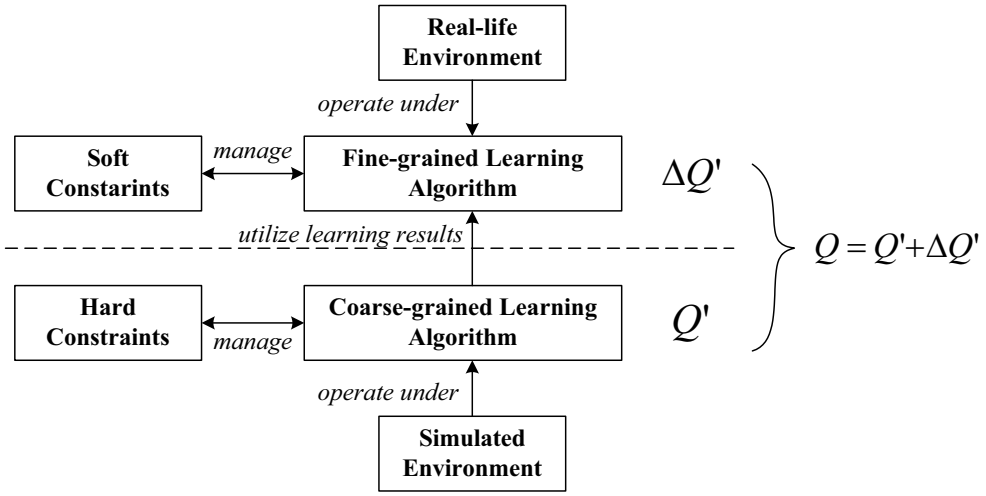

Figure 4.4: The relations between the two learning algorithms.

4.4 Coarse-grained Learning Algorithm

In the coarse-grained learning algorithm, the partition fluents of each row of the table are divided into three groups (ref. Figure 4.3), denoted by D_s, D'_s, and A_f respectively. D_s refers to the set of fluents characterizing the current system state S. A_f is the set of fluents representing the specific domain method M executable in S. The degrees to which the relevant hard system constraints are satisfied by method M are included in A_f as individual fluent members. D'_s denotes the fluents characterizing the expected system state S' when the method M finishes. As these fluents establish a specific system partition E on the system states and

domain methods, the coarse-grained learning algorithm is used to estimate the corresponding aggregated qualities as defined in equation (4.6). The quality of performing each method can be written as:

$$Q(S, S', M) = R(S, S') + \alpha \cdot \underset{(S'', M')}{Avg} \ (Q(S', S'', M')) \tag{4.13}$$

To simplify the description, equation (4.13) has omitted the differences among the discount rates at the various system states, and the discount rate is denoted uniformly by α. Assume initially that the application domain satisfies assumption A^*, and the system partition E satisfies assumptions A1–A4. Assumption A4 can be omitted in practice. As shown later, the coarse-grained learning algorithm is able to handle the violation of assumptions A^*. For now, let $\Psi(B^S, B^{S'}, B^M)$ denote the set of triplets (S, S', M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$. The definition of the aggregated quality function is expanded as

$$\begin{aligned} Q^*(B^S, B^{S'}, B^M) &= \underset{(S,S',M)\in\Psi(B^S,B^{S'},B^M)}{Avg} \ (Q^*(S, S', M)) \\ &= \underset{(S,S',M)\in\Psi(B^S,B^{S'},B^M)}{Avg} \left[R(S, S') + \alpha \cdot \underset{(S'', M')}{Avg} \ (Q^*(S', S'', M')) \right] \\ &= \frac{\sum_{(S,S',M)\in\Psi(B^S,B^{S'},B^M)} R(S, S')}{\left\|\Psi(B^S, B^{S'}, B^M)\right\|} + \alpha \cdot \frac{\eta(B^{S'})}{\left\|\Psi(B^S, B^{S'}, B^M)\right\|} \end{aligned}$$

where

$$\eta(B^{S'}) = \sum_{S'\in B^{S'}} \left[\underset{(S'', M')}{Avg} \ (Q^*(S', S'', M')) \right]$$

For each $S' \in B^{S'}$, the number of triplets (S', S'', M') such that $S' \xrightarrow{M''} S''$ is the same (due to assumption A3), and is represented as $\varphi(B^{S'})$. Then $\eta(B^{S'})$ can be written as

$$\begin{aligned} \eta(B^{S'}) &= \sum_{S'\in B^{S'}} \left\{ \frac{\sum_{(S'',M')} Q^*(S', S'', M')}{\left\|\varphi(B^{S'})\right\|} \right\} \\ &= \frac{1}{\varphi(B^{S'})} \cdot \sum_{S'\in B^{S'}} \sum_{(S'',M')} Q^*(S', S'', M') \end{aligned}$$

Let $\omega(B^{S'})$ denote the set of triplets $(B^{S'}, B^{S''}, B^{M'})$ for which $B^{S'} \xrightarrow{B^{M'}} B^{S''}$ is valid. By assumptions A1–A4,

$$\eta(B^{S'}) = \frac{1}{\varphi(B^{S'})} \cdot \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \tag{4.14}$$

The equation for aggregated quality functions can now be written as

$$Q^*(B^S, B^{S'}, B^M) = \underset{(S,S',M)\in\Psi(B^S,B^{S'},B^M)}{Avg} R(S,S') + \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \cdot \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \quad (4.15)$$

Equation (4.15) is the *Bellman equation* [23] for the coarse-grained learning problem. The corresponding updating rule of the algorithm therefore is

$$Q(B^S, B^{S'}, B^M) \leftarrow (1 - \frac{1}{n(B^S, B^{S'}, B^M)}) \cdot Q(B^S, B^{S'}, B^M) + \frac{1}{n(B^S, B^{S'}, B^M)} \left[R(S,S') + \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \cdot \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left(\left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q(B^{S'}, B^{S''}, B^{M'}) \right) \right] \quad (4.16)$$

An agent will carry out the learning process determined by the above updating rule when it finishes executing a method $M \in B^M$. The function $n(B^S, B^{S'}, B^M)$ returns the number of times $Q(B^S,B^{S'},B^M)$ (i.e., the estimation of Q^*) has been updated (Ref. equation (4.16)). The inverse of this function serves as the learning rate $\gamma(B^S, B^{S'}, B^M)$ of the updating rule. This updating rule can be modified to cover the case where assumption A^* is violated. Specifically, the *quality* of performing a method M is re-defined as

$$Q^*(S, S', M) = \sum_{S''\in\{S''|Pr(S''|S,S',M)>0\}} Pr(S''\,|S, S', M\,) \cdot \left[R(S,S'') + \alpha \cdot \underset{(S''',M')}{Avg} Q^*(S'', S''', M') \right]$$

where $Pr(S''\,|S, S', M\,)$ is the probability of reaching the system state S'' when M is finished. By performing similar mathematical manipulations to the one in the previous discussions, the updating rule for the modified coarse-grained learning algorithm is obtained as follows:

$$Q(B^S, B^{S'}, B^M) \leftarrow (1 - \gamma_t) \cdot Q(B^S, B^{S'}, B^M) + \gamma_t \cdot R(S,S'') + \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S''}) \cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \cdot \sum_{(B^{S''},B^{S'''},B^{M'})\in\omega(B^{S''})} \left\|\Psi(B^{S''}, B^{S'''}, B^{M'})\right\| Q(B^{S''}, B^{S'''}, B^{M'}) \quad (4.17)$$

An important premise of adopting this updating rule is that the agents should know *a priori* the cardinality of each set $\Psi(B^S, B^{S'}, B^M)$. Unfortunately, this is often an impractical requirement for many application domains. In order to obtain a practically applicable updating rule, the updating rule in equation (4.17) is further simplified as follows:

$$Q(B^S, B^{S'}, B^M) \leftarrow (1-\gamma_t) \cdot Q(B^S, B^{S'}, B^M) + \gamma_t \cdot \left[R(S, S'') + \gamma_t \cdot \alpha \cdot \frac{\sum_{(B^{S''}, B^{S'''}, B^{M'}) \in \omega(B^{S''})} Q(B^{S''}, B^{S'''}, B^{M'})}{\left\| \omega(B^{S''}) \right\|} \right] \tag{4.18}$$

This *is* the updating rule adopted in the coarse-grained learning algorithm. The algorithm in procedural form is presented in Figure 4.5. It should be noted that this updating rule cannot guarantee that the learning algorithm converges towards the desired estimation of *qualities*, Q^*, due to the removal of the terms $\left\| \Psi(B^S, B^{S'}, B^M) \right\|$ and $\varphi(B^S)$. Nevertheless, as shown in Section 4.6, the algorithm can still converge towards a sub-optimal learning result. This property is much more desirable than that of an algorithm which may achieve optimality but is with highly stringent requirements, especially for the real-world applications with complex dynamics.

Initialize $Q(B^s, B^{s'}, B^M)$

Repeat (for each Simulated Running of the System)

Determine the initial system state S^0

Repeat (for each Learning Step)

(1) Determine the current system state S

(2) Choose a method M to perform using the learning policy derived from the current learning results.

(3) Perform the method M until it is completed. Record the reward R incurred and the afterwards system state S'.

(4) Update the learning results using equation (27).

Until the system terminate

Figure 4.5: The coarse-grained learning algorithm.

4.5 Fine-grained Learning Algorithm

Different from the coarse-grained learning algorithm, the fine-grained learning algorithm estimates the expected quality changes when the execution of each

domain method is under the restrictions of soft system constraints. To characterize the specific *soft constraints* pertinent to each candidate method M, a so-called *method relation graph*, G_M, is constructed for M. Similar with the *soft relation graphs*, G_M is represented as a tuple of four components, $G_M = (V, E, \epsilon, \beta)$. The function β maps each edge $e \in E$ to a soft relation. The function ϵ assigns to each $v \in V$ either a meta-method $\overline{M}^v$ or a domain method M^v. Particularly, there must be one vertex $u \in V$ such that $\epsilon(u) = M$.

For any method M, the corresponding method relation graph G_M is constructed directly from the soft relation graph g. The construction procedure is summarized in Figure 4.6. A soft relation graph in a FSTS model is typically comprised of several (e.g. k) disconnected subgraphs. The similarity degree between each subgraph and the method relation graph of M, G_M, is consequently considered as forming one individual domain fluent. A group of fluents $S_g = \{S_{g1}, S_{g2}, \ldots, S_{gk}\}$ is used to characterize the soft relations pertinent to M, where S_{gn} denotes the similarity degree between the n-th subgraph of the soft relation graph and the method relation graph. The notion of the graph similarity measure is adapted from the concept of *error-tolerant graph matching* (etgm) [44].

S1 Initiate G_M with a graph that contains no vertices and edges.

S2 *If* M belongs to meta-method $\overline{M}^u$ denoted by vertex u of the soft relation graph g, *then* add u to G_M and set the label of u to M. *Otherwise*, goto step **S7**.

S3 Add all vertices in g that is directly connected with vertices of G_M. The label of each new dded vertex is identical with that in graph g.

S4 Link the new added vertices with those already in G_M by the edges that apear in graph g. The label of each new added edge is identical with that in graph g.

S5 *If* there exist other vertices in graph g that are directly connected with the vertices currently included in G_M, *then* goto step **S3**. *Otherwise*, goto step **S6**.

S6 *For* each vertex v in G_M, *if* the label of v is a meta-method $\overline{M}^v$, and there exists one method $M' \in \overline{M}^v$, such that M' has been performed or is being performed, *then* replace the label of v with M'.

S7 The resulting graph is the method relation graph of M, G_M.

Figure 4.6: The procedure for constructing the method relation graph G_M of a method M.

This group of fluents S_g belongs to A_f. They refine the partition E adopted by the coarse-grained learning algorithm. For each method block B^M in E, there may be multiple method blocks in the refined partition. However, as soft constraints take a significant role in the fine-grained learning algorithm, the fluent group S_g is represented explicitly in the following discussions. Let Is_g denote any valid combination of fluent values of S_g. And let G_M be any particular *method*

relation graph characterized by Is_g, $G_M \in Is_g$. The expected *quality changes* of performing a method M due to the existence of G_M is defined as

$$\begin{aligned}&\Delta Q^*(S, S', M, Is_g) = \\ &\underset{G_M \in Is_g}{Avg} \Big(Q^*(S, S', M, G_M) - Q^*(S, S', \hat{M})\Big)\end{aligned} \tag{4.19}$$

This definition is based upon the assumption that any expected change of system states by performing method M under the existence of soft relations G_M is achievable by performing another method $\hat{M}$ when G_M does not exist. By exploiting assumptions A1–A4, the expected *quality changes* in the case of aggregated systems are

$$\begin{aligned}&\Delta Q^*(B^S, B^{S'}, B^M, Is_g) = \\ &Q^*(B^S, B^{S'}, B^M, Is_g) - Q^*(B^S, B^{S'}, B^M)\end{aligned} \tag{4.20}$$

where

$$\begin{aligned}&Q^*(B^S, B^{S'}, B^M, Is_g) = \\ &\underset{G_M \in Is_g}{Avg} \quad \underset{(S,S',M) \in \Psi(B^S, B^{S'}, B^M, G_M)}{Avg} \quad Q^*(S, S', M, G_M)\end{aligned} \tag{4.21}$$

$\Psi(B^S, B^{S'}, B^M, G_M)$ denotes the set of tuples (S, S', M, G_M) such that (1) the condition $S \xrightarrow{M} S'$ is valid; and (2) the *soft constraints* relevant to the method M is characterized by G_M. Assume that $Q^*(B^S, B^{S'}, B^M)$, which is estimated by the coarse-grained learning algorithm, is known *a priori*. It is therefore obvious that estimating $Q^*(B^S, B^{S'}, B^M, Is_g)$ and $\Delta Q^*(B^S, B^{S'}, B^M, Is_g)$ are actually identical. Consider first the case of estimating $Q^*(B^S, B^{S'}, B^M, Is_g)$.

Notice that each $Q^*(S, S', M, G_M)$, $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $G_M \in Is_g$, can be regarded as one specific estimation of $Q^*(B^S, B^{S'}, B^M, Is_g)$ with a corresponding error term w. Specifically, by adopting the Robbins-Monro algorithm [65, 227], the corresponding updating rule for estimating Q^* is

$$\begin{aligned}Q(B^S, B^{S'}, B^M, Is_g) \quad \leftarrow \quad &(1 - \gamma_t) \cdot Q(B^S, B^{S'}, B^M, Is_g) \\ &+\gamma_t \cdot Q^*(S, S', M, G_M)\end{aligned} \tag{4.22}$$

Now consider the estimation of $Q^*(S, S', M, G_M)$. Suppose that a method M starts to be performed at a system state S, and the system state is changed to S' when M finishes. The corresponding *temporal difference* $d_{S,S'}$ is

$$\begin{aligned}d_{S,S'} = R(S, S') + \alpha \cdot &\underset{(S', S'', M', G'_M)}{Avg} (Q(S', S'', M', G'_M)) \\ &-Q(S, S', M, G_M)\end{aligned} \tag{4.23}$$

where $Q(S, S', M, G_M)$ is the current estimation of $Q^*(S, S', M, G_M)$. The fine-grained updating rule for estimating $Q^*(S, S', M, G_M)$ is then constructed as

$$\begin{array}{c} Q(S, S', M, G_M) \leftarrow Q(S, S', M, G_M) + \gamma_m \\ \cdot Z_m(S, S', M, G_M) \cdot d_{S_m, S_{m+1}} \end{array} \tag{4.24}$$

where Z_m is the *eligibility coefficient*, and is defined as

$$Z_m(S, S', M, G_M) = \begin{cases} \alpha\lambda Z_{m-1}(S, S', M, G_M), & (S_m, S_{m+1}, M_m, G_{Mm}) \\ & \neq (S, S', M, G_M) \\ \alpha\lambda Z_{m-1}(S, S', M, G_M) & (S_m, S_{m+1}, M_m, G_{Mm}) \\ +1, & = (S, S', M, G_M) \end{cases} \tag{4.25}$$

By extending this algorithm for the aggregated systems, and incorporating it into equation (4.22), the updating rule for estimating $Q^*(B^S, B^{S'}, B^M, Is_g)$ is obtained as follows:

$$\begin{array}{c} Q(B^S, B^{S'}, B^M, Is_g) \leftarrow Q(B^S, B^{S'}, B^M, Is_g) \\ +\gamma_m \cdot Z_m(B^S, B^{S'}, B^M, Is_g) \cdot d_{S_m, S_{m+1}} \end{array} \tag{4.26}$$

where $Z_m(B^S, B^{S'}, B^M, Is_g)$ is defined similarly to $Z_m(S, S', M, G_M)$. Following this updating rule and the assumption that $Q^*(B^S, B^{S'}, B^M)$ is known *a priori*, the corresponding updating rule to estimate ΔQ^* is

$$\begin{array}{c} \Delta Q(B^S, B^{S'}, B^M, Is_g) \leftarrow \Delta Q(B^S, B^{S'}, B^M, Is_g) \\ +\gamma_m \cdot Z_m(B^S, B^{S'}, B^M, Is_g) \cdot d_{S_m, S_{m+1}} \end{array} \tag{4.27}$$

For the aggregated systems, the evaluation of the temporal difference $d_{S_m, S_{m+1}}$ in equation (4.23) is slightly altered as follows:

$$\begin{aligned} d_{S_m, S_{m+1}} \quad = \quad & R(S_m, S_{m+1}) + \alpha \cdot \\ & \underset{(B^{S^{m+2}}, B^{M^{m+1}}, {Is_g}^{m+1})}{Avg} \left(Q(B^{S^{m+1}}, B^{S^{m+2}}, B^{M^{m+1}}, {Is_g}^{m+1}) \right) \\ & - Q(B^{S^m}, B^{S^{m+1}}, B^{M^m}, {Is_g}^m) \end{aligned} \tag{4.28}$$

where B^{S^m} refers to the state block such that $S_m \in B^{S^m}$. Similar conventions are used for the notations B^{M^m} and ${Is_g}^m$.

Basically, the fine-grained learning algorithm follows an updating rule determined by equation (4.27). However, the actual form needs to be modified in order to work efficiently with the Takagi-Sugeno fuzzy rule base, which is used to maintain the learning results. Particularly, suppose that there are N fuzzy rules in the rule base. Use b_k to denote the output of the k-th fuzzy rule. Let

$$\phi_k(S, S', M, G_M) = \frac{\mu_k(S, S', M, G_M)}{\sum_{i=1}^{N} \mu_i(S, S', M, G_M)} \tag{4.29}$$

where $\mu_k(\bullet)$ refers to the similarity degree of any tuple (S, S', M, G_M) with respect to the preconditions of the k-th fuzzy rule of the rule base. Since the fuzzy rule base serves only as an approximation of ΔQ^*, the actual number of fuzzy rules is adjustable. The system designers can choose a proper number such that the fine-grained learning algorithm does not impose too much computational requirements. Denote ϕ to be the N-*vector* of ϕ_k. The output of the Takagi-Sugeno fuzzy system (an estimation of $\Delta Q^*(B^S, B^{S'}, B^M, Is_g)$) is:

$$\Delta Q(B^S, B^{S'}, B^M, Is_g) = \sum_{k=1}^{N} \phi_k(S, S', M, G_M) \cdot b_k \tag{4.30}$$

where (S, S', M, G_M) can be any tuple that belongs to $\Psi(B^S, B^{S'}, B^M, Is_g)$. Equation (4.30) shows that the value of ΔQ is actually determined by the outputs (b_k) of each fuzzy rule. Consequently, without updating the estimation of ΔQ, the updating rule of the fine-grained learning algorithm adjusts the value of each b_k, instead. Let the *output vector* $\vec{b}$ be the vector $(b_1, b_2, \ldots, b_N)^T$. According to [29], $\vec{b}$ can be actually updated through

$$\vec{b} \leftarrow \vec{b} + \gamma_m \cdot Z_m \cdot d_{S_m, S_{m+1}} \tag{4.31}$$

where Z_m is defined as

$$Z_m = \sum_{i=1}^{m} (\alpha \cdot \lambda)^{m-1} \cdot \phi(S_i, S_{i+1}, M_i, G_{Mi}) \tag{4.32}$$

This *is* the updating rule that is used in the fine-grained learning algorithm. The updating procedure is carried out each time when certain domain method M has been finished. When applying this fine-grained learning algorithm, *two* learning strategies are applicable.

In the *fixed policy* learning strategy, the decision policy is derived directly from the coarse-grained learning algorithm. At each decision point, agents will most probably perform the methods that achieve the highest quality Q' estimated by the coarse-grained learning algorithm. The estimated quality changes $\Delta Q'$ are incorporated into each agents' decision-making procedure only after the conclusion of the fine-grained learning algorithm. From the perspective of policy iterations [29], this strategy improves only once the decision policies which are determined by the coarse-grained learning algorithm. Adopting this strategy, the convergence of the fine-grained learning algorithm is ensured (Ref. Section 4.6). However, since this strategy improves policies only once, the expected improvement of the system performance may not justify the extra computational consumption introduced by the fine-grained learning algorithm. A possible approach to this concern is *optimistic policy iterations*, where the fine-grained learning algorithm is comprised

of a sequence of learning *phases*. Within any particular phase, the expected quality changes $\Delta Q'$ under a fixed policy π is estimated. The policy π is replaced by another policy π' when the algorithm progresses from one phase to another. Policy π' is considered as an improvement of policy π since the method that achieves the highest $Q' + \Delta Q$ in π will most probably be selected by π'.

At a first glance, optimistic policy iterations may improve further the system performance achievable by the first learning strategy described above. Nevertheless, as investigated in [29], the fine-grained learning algorithm may not be convergent with this approach. In order to attain certain convergence guarantees, the *restricted policy iteration* learning strategy is proposed where the policies adopted by each agent are continually altered. The overall learning process is similar to the optimistic policy iterations, but they differ in the following two aspects:

1. the adjustment of the outputs of the fuzzy rules ($\vec{b}$) in the restricted policy iteration cannot exceed a predetermined bound;

2. for any two consecutive learning phases, the differences between the two groups of fuzzy rule outputs are bounded and decreasing in general.

This restricted learning strategy is shown in Figure 4.7. During each learning phase, two groups of fuzzy rule outputs will be identified as ${}^1\vec{b}$ and ${}^2\vec{b}$. The first group ${}^1\vec{b}$ approximates the quality changes $\Delta Q'$ of performing each method under the current learning policy π. The agents use the second group ${}^2\vec{b}$ derived from ${}^1\vec{b}$ to establish another policy π' which is to be adopted during the next learning phase. The policy π' will most probably choose the method that achieves the highest $Q' + \Delta Q'$ under ${}^2\vec{b}$.

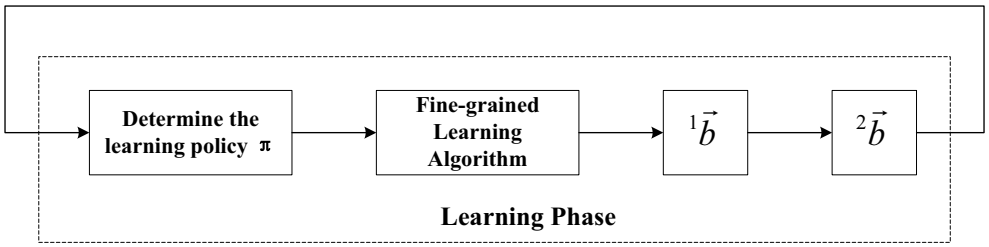

Figure 4.7: The restricted policy iteration process.

During each learning phase, the updating process follows equation (4.31) to produce the vector ${}^1\vec{b}$. Different from optimistic policy iterations where ${}^1\vec{b}$ is used to construct directly the policy π' of the next learning phase, the vector ${}^2\vec{b}$ is used. The procedure that derives ${}^2\vec{b}$ from ${}^1\vec{b}$ demonstrates the above two aspects that distinguish the restricted policy iteration from the optimistic policy iterations. To make the two aspects explicit, several regulations have been proposed. First, the

initial fuzzy rule outputs ${}^0\vec{b}$ are set to be a zero vector $\vec{0}$. In order to respect the first aspect, a *trapping function* $trap(\vec{b})$ is used to map any vector $\vec{b}$ which is far from ${}^0\vec{b}$ (i.e. $\left\|\vec{b}\right\|$ is large) to another vector $\vec{b}'$, which is much more closer to ${}^0\vec{b}$. An example trapping function is given in equation (4.33).

$$\forall b_i \ of \ \vec{b}, b_i' = \frac{b_i}{\left\|\vec{b}\right\|} \cdot trap^* \left(\left\|\vec{b}\right\|\right) \tag{4.33}$$

where $\left\|\vec{b}\right\|$ can be any vector norm of $\vec{b}$, and the function $trap^*(\bullet)$ is a one-dimensional real function (Ref. Figure 4.8).

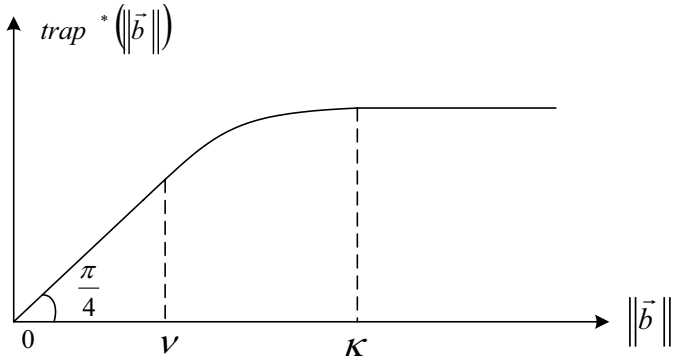

Figure 4.8: The general shape of the function $trap^*(\bullet)$.

In Figure 4.8, ν and κ are two variables. They jointly control the shape and bound of the corresponding trapping function $trap(\vec{b})$. κ serves as the function bound, that is, the norm of any vector produced by the trapping function cannot exceed κ. Additionally, if $\left\|\vec{b}\right\|$ is smaller than ν, the trapping function maps $\vec{b}$ to itself.

Second, a *step-weight* δ, which adjusts the differences between the vectors ${}^2\vec{b}$ of any two consecutive learning phases, is introduced. Let ${}^2\vec{b}'$ denote the output vector that establishes the decision policy π of the current learning phase, and ${}^1\vec{b}$ denote the output vector identified by the fine-grained learning process of the same phase. The vector ${}^2\vec{b}$, which is used to determine the policy π' of the next learning phase, is obtained from the H function

$$H({}^2\vec{b}', {}^1\vec{b}) = \delta \cdot {}^1\vec{b}$$

Since vector ${}^1\vec{b}$ is actually determined by vector ${}^2\vec{b}'$, the H function can be re-written as $H({}^2\vec{b}')$. In order to ensure that the differences between ${}^2\vec{b}'$ and ${}^2\vec{b}$ are decreasing in general, an updating rule given in equation (4.34) is adopted to determine the vector ${}^2\vec{b}$ at each learning phase. It is frequently used in stochastic approximation algorithms.

$$^2\vec{b} \leftarrow trap\left((1-\gamma_t)\cdot^2 \vec{b} + \gamma_t \cdot H(^2\vec{b})\right) \tag{4.34}$$

Note that γ_t in equation (4.34) is a positive real number and stands for a phase-level learning rate of the fine-grained learning algorithm. The subscript t indicates the number of learning phases completed previously. The value of γ_t decreases as t increases, and satisfies the typical requirements of learning rates. For the detailed discussion on it, see the next Section where the fine-grained learning algorithm is proved to converge under the restricted policy iterations by choosing a proper step-weight δ.

4.6 Theoretical Analysis of the Learning Algorithms

This section is dedicated to the theoretical analysis focusing on the convergence of the learning algorithms described in Sections 4.4 and 4.5. Four convergence results are presented. The first two results are derived for the coarse-grained learning algorithm. They show that the algorithm adopting the updating rule provided in either equation (4.16) or equation (4.18) is convergent under proper conditions. The third and fourth theoretical results are derived for the fine-grained learning algorithm. They indicate that the algorithm is convergent for both learning strategies described in Section 4.5. However, the convergence results require system designers to properly select both the discount rate α and the step-weight δ.

Before presenting the four theoretical results, the two theorems from [264], which form the basis of the proof of the first convergence result, will be introduced. Let $T : B \rightarrow B$ be an arbitrary operator, where B is the space of uniformly bounded functions over a given set. Let $\Gamma = (T_0, T_1, \ldots T_t, \ldots)$ be a sequence of operators, and $T_t : B \times B \rightarrow B$. Let $F \subseteq B$ be a subset of B and let $F_0 : F \rightarrow 2^B$ be a mapping that associates subsets of B with the elements of F. If, for all $f \in F$ and all $m_0 \in F_0(f)$, the sequence generated by the recursion $m_{t+1} = T_t(m_t, f)$ converges to Tf, then Γ is said to approximate T for initial values from $F_0(f)$ and on the set $F \subseteq B$. Further, the subset $F \subseteq B$ is invariant under $T' : B \times B \rightarrow B$ if, for all $f, g \in F$, $T'(f, g) \in F$. If Γ is an operator sequence as above, then F is said to be invariant under Γ if for all $i \geq 0$, F is invariant under T_i. The two theorems from [264] are presented as Theorem 3 and Theorem 4.

Theorem 3 *Let χ be an arbitrary set and assume that B is the space of bounded functions over χ, $B(\chi)$. $T : B(\chi) \rightarrow B(\chi)$. Let v^* be a fixed point of T, and $\Gamma = (T_0, T_1, \ldots T_t, \ldots)$ approximate T at v^* for initial values from $F_0(v^*)$. Assume that F_0 is invariant under Γ. Let $V_0 \in F_0(v^*)$, and define $V_{t+1} = T_t(V_t, V_t)$. If*

there exist random functions $0 \leq F_t(x) \leq 1$ *and* $0 \leq G(x) \leq 1$ *satisfying the conditions below with probability 1, then* V_t *converges to* v^* *with probability 1:*

1. *For all* U_1 *and* U_2, $U_1, U_2 \in F_0$, *and for all* $x \in \chi$, $|T_t(U_1, v^*)(x) - T_t(U_2, v^*)(x)| \leq G_t(x) \cdot |U_1(x) - U_2(x)|$

2. *For all* U *and* V, $U, V \in F_0$, *and all* $x \in \chi$, $|T_t(U, v^*)(x) - T_t(U, V)(x)| \leq F_t(x) \cdot (\|v^* - V\| + \lambda_t)$ *where* $\lambda_t \to 0$ *with probability 1 as* $t \to \infty$.

3. *For all* $k > 0$, $\prod_{t=k}^{n} G_t(x)$ *converges to 0 uniformly in* x *as* $n \to \infty$.

4. *There exists* $0 \leq \gamma < 1$ *such that for all* $x \in \chi$ *and large enough* t, $F_t(x) \leq \gamma \cdot (1 - G_t(x))$

Theorem 4 *Considering the updating process* $Q_{t+1} = (1 - \gamma_t) \cdot Q_t + \gamma_t w_t$, *if* $\sum_{t=1}^{\infty} \gamma_t = \infty$, $\sum_{t=1}^{\infty} (\gamma_t)^2 < C < \infty$, $E[w_t | h_t, \gamma_t] = A$, *and* $E[w_t^2 | h_t] < B < \infty$, *where* h_t *is the system history at time* t, *and* $B, C > 0$, *then the updating process converges to* A *with probability 1.*

Now the coarse-grained learning algorithm governed by equation (4.16) can be proved to converge.

Proposition 1 *Suppose that the updating process of the coarse-grained learning algorithm follows equation (4.16). The algorithm converges to the aggregated quality function* $Q(B^S, B^{S'}, B^M)$ *defined in equation (4.6) if the following three conditions are met, and the initial learning rate* γ_0 *is small enough*[1]*:*

1. *The partition of the system states and domain methods satisfies assumptions A1–A4.*

2. *The application domain satisfies assumption* A^*.

3. *The learning rate* γ_t *satisfies the conditions:*

$$\sum_{t=0}^{\infty} \gamma_t = \infty, \sum_{t=0}^{\infty} \gamma_t^2 < C < \infty$$

where C *is a positive real number.*

Proof: Examining the updating rule of the coarse-grained learning algorithm, a sequence of random operators, $\tilde{\Gamma} = (\tilde{T_1}, \tilde{T_2}, \ldots, \tilde{T_n}, \ldots)$, can be identified. Each operator takes the form:

[1] What exactly is meant by "small enough" will become clear in the proof of this proposition

$$\tilde{T}_t(Q', Q) = (1-\gamma_t)\cdot Q' + \gamma_t \cdot \left[R + \alpha \cdot \frac{1}{\varphi \cdot \|\Psi\|} \cdot \sum \|\Psi'\| \cdot Q\right]$$

For a specific case where the quality of the triplet (B^S, $B^{S'}$, B^M) is to be updated, Q and Q' in the above equation can be detailed as $Q(B^{S'}, B^{S''}, B^{M'})$ and $Q'(B^S, B^{S'}, B^M)$, respectively. As a consequence, φ refers to $\varphi(B^{S'})$, R denotes $R(S, S')$ such that $S \in B^S$, $S' \in B^{S'}$. Ψ and Ψ' stand for $\Psi(B^S, B^{S'}, B^M)$ and $\Psi(B^{S'}, B^{S''}, B^{M'})$. And the summation is around all potential triplets $(B^{S'}, B^{S''}, B^{M'})$ such that $(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})$.

Now consider a special updating process using the above sequence of operators. That is, $Q'_{t+1} = \tilde{T}_t(Q'_t, Q^*)$. If

$$\begin{aligned} w_t \quad = \quad & R(S, S') + \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \\ & \sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \left(\|\Psi(B^{S'}, B^{S''}, B^{M'})\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \right) \end{aligned}$$

By Theorem 4, the updating sequence $Q'_{t+1} = \tilde{T}_t(Q'_t, Q^*)$ converges to

$$E\left[w_t | h_t, \gamma_t\right]$$

Notice the following three points that apply to this updating process:

1. Because the updating process is carried out within a simulated environment, the initial system state can be selected at random. Thus, the probability of choosing each state S in the same state block B^S is identical.

2. For any triplet (S, S', M) satisfying the condition: $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$, the corresponding quality given by the aggregated quality function is equivalent. Therefore, with any fixed aggregated quality function, the learning policy (i.e. the policy adopted during the learning process) will give each triplet with the same condition an equal probability of being observed.

3. Due to the one-to-one correspondence among the system states in different state blocks, for any initial state that belongs to the same state block B^S and any learning policy π, the probability for the afterwards system state to lie within another state block $B^{S'}$ is the same.

Therefore, the probability of observing each triplet (S, S', M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \xrightarrow{M} S'$, is identical and equals to:

$$\frac{1}{\|\Psi(B^S, B^{S'}, B^M)\|}$$

Then, Q'_t will converge to

$$Avg\left\{\begin{array}{c} R(S,S') + \alpha \cdot \dfrac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \\ \displaystyle\sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \left(\|\Psi(B^{S'}, B^{S''}, B^{M'})\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \right) \end{array}\right\}$$
$$= Q^*(B^S, B^{S'}, B^M)$$

It implies that the sequence of operators $\tilde{\Gamma}$ approximates the operator $\tilde{T}$ at Q^* such that $\tilde{T}Q^* = Q^*$.

Given the fact that the updating process $Q'_{t+1} = \tilde{T}_t(Q'_t, Q^*)$ converges to Q^*, consider now the process $Q'_{t+1} = \tilde{T}_t(Q'_t, Q'_t)$, which is the updating rule of the coarse-grained learning algorithm. In the following, this process is shown to converge to Q^* by verifying that the four conditions of Theorem 3 are satisfied:

1. $|\tilde{T}_t(U_1, Q^*) - \tilde{(U_2, Q^*)}| \leq (1-\gamma_t)|U_1 - U_2|$
   Let $G_t = 1 - \gamma_t$, then $|\tilde{T}_t(U_1, Q^*) - \tilde{(U_2, Q^*)}| \leq G_t \cdot |U_1 - U_2|$
   Condition 1 is therefore satisfied.

2.
$$\begin{aligned} & \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \\ & \left| \sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \|\Psi(B^{S'}, B^{S''}, B^{M'})\| \cdot \left(\begin{array}{c} Q^*(B^{S'}, B^{S''}, B^{M'}) - \\ Q(B^{S'}, B^{S''}, B^{M'}) \end{array} \right) \right| \\ \leq \; & \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \\ & \sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \|\Psi(B^{S'}, B^{S''}, B^{M'})\| \cdot \left| \begin{array}{c} Q^*(B^{S'}, B^{S''}, B^{M'}) - \\ Q(B^{S'}, B^{S''}, B^{M'}) \end{array} \right| \\ \leq \; & \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \|\omega(B^{S'})\| \cdot \\ & \Psi'' \cdot \|Q^*(B^{S'}, B^{S''}, B^{M'}) - Q(B^{S'}, B^{S''}, B^{M'})\|_\infty \end{aligned}$$

   $\|\cdot\|_\infty$ is the maximum norm. Ψ'' is defined as

$$\Psi'' = \max_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \|\Psi(B^{S'}, B^{S''}, B^{M'})\|$$

   Introducing a constant Con

$$Con = \max_{(B^S, B^{S'}, B^M)} \left(\alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \|\omega(B^{S'})\| \cdot \Psi'' \right)$$

   and defining the function F_t as

$$F_t = \gamma_t \cdot Con$$

the inequality

$$|\tilde{T}_t(U, Q^*) - \tilde{T}_t(U, Q)| \leq F_t \cdot \|Q^* - Q\|_\infty$$

is obtained under the assumption that $Con < 1$ or $F_t|_{t=0} < 1$. Condition 2 of theorem 3 is therefore satisfied.

Note that for many applications, it is convenient to have an assumption of $Con < 1$, which can be achieved by selecting a proper discount value α or by adjusting the *initial learning rate*, $\gamma_t|_{t=0}$, so that $\gamma_0 \cdot Con < 1$.

3. As described previously, define γ_t as $\gamma_t = \frac{1}{t}$, G_t is then written as

$$G_t = \frac{t-1}{t}, \text{ so that}$$
$$\prod_{t=k}^{n} = \frac{k-1}{k} \cdot \frac{k}{k+1} \cdot \cdots \cdot \frac{n-1}{n} = \frac{k-1}{n}$$
$$lim_{n\to\infty} \left(\prod_{t=k}^{n} G_t \right) = 0$$

Condition 3 is therefore satisfied.

4. Since $1 - G_t = \gamma_t$, the condition 4 of theorem 1 consequently implies that $Con < \gamma$, where γ is another positive constant that is lower than 1. Due to the assumption that $Con < 1$, this condition is trivially satisfied.

The four conditions of theorem 3 under proper assumptions are all met, and the coarse-grained learning algorithm therefore converges to the aggregated quality function defined in equation (4.6).

Q.E.D

Next the coarse-grained learning algorithm governed by equation (4.18) is shown to converge.

Proposition 2 *If the updating process of the coarse-grained learning algorithm follows the equation (4.18), the algorithm converges.*

Proof: To show that the updating process described by equation (4.18) finally converges, it is suffice to consider the update carried out for any triplet $(B^S, B^{S'}, B^M)$. First, equation (4.18) is re-written as

$$Q(B^S, B^{S'}, B^M) \leftarrow (1 - \gamma_t) \cdot Q(B^S, B^{S'}, B^M) + \gamma_t \cdot \Big[(HQ)(B^S, B^{S'}, B^M) + w_t(B^S, B^{S'}, B^M) \Big]$$

where (HQ) is defined as

$$(HQ)(B^S, B^{S'}, B^M) = E\left[\begin{array}{c} R(S,S'')|_{S''\in B^{S''}} + \\ \alpha \cdot \dfrac{\sum\limits_{(B^{S''},B^{S'''},B^{M'})\in\omega(B^{S''})} Q(B^{S''}, B^{S'''}, B^{M'})}{\|\omega(B^{S''})\|} \end{array}\right]$$

w_t refers to the noises inherent in the updating process, and is represented as

$$w_t(B^S, B^{S'}, B^M) = R(S,S'')|_{S''\in B^{S''}} + \alpha \cdot \underset{(B^{S''},B^{S'''},B^{M'})\in\omega(B^{S''})}{Avg} Q(B^{S''}, B^{S'''}, B^{M'}) - (HQ)(B^S, B^{S'}, B^M)$$

It can be verified that $E[w_t|h_t, \gamma_t] = 0$. Since w_t defined above is bounded, there exist positive constants A and B, such that

$$E[w_t^2|h_t, \gamma_t] \leq A + B \cdot \|Q_t\|_2^2$$

In addition, (HQ) is a contraction mapping. For two different estimation of qualities Q and $\bar{Q}$,

$$\begin{aligned} &|(HQ)(B^S, B^{S'}, B^M) - (H\bar{Q})(B^S, B^{S'}, B^M)| \\ &\leq \alpha \cdot \sum_{B^{S''}} \left[\begin{array}{c} Pr(B^{S''}|B^S, B^{S'}, B^M)\cdot \\ max \left| \begin{array}{c} Q(B^{S''}, B^{S'''}, B^{M'}) - \\ \bar{Q}(B^{S''}, B^{S'''}, B^{M'}) \end{array} \right| \end{array}\right] \\ &\leq \alpha \cdot max \left|Q(B^{S''}, B^{S'''}, B^{M'}) - \bar{Q}(B^{S''}, B^{S'''}, B^{M'})\right| \\ &\leq \alpha \cdot \|Q - \bar{Q}\|_\infty \end{aligned} \tag{4.35}$$

where max is taken over all triplets $(B^{S''}, B^{S'''}, B^{M'}) \in \omega(Id_s'')$, and $\|\cdot\|_\infty$ is the maximum vector norm. Taking the maximum over the left side of inequality (4.35), then

$$\|(HQ) - (H\bar{Q})\|_\infty \leq \alpha \cdot \|Q - \bar{Q}\|_\infty$$

where the discount rate $\alpha < 1$, since the mapping (HQ) is contractive. Besides, the noise term w_t is bounded and has zero as its expected value. By utilizing traditional stochastic approximation theories (i.e. *Proposition 4.4* in [29]), the updating process described by equation (4.18) converges to $\bar{Q}^*$, which is the solution of the equation of the form $(H\bar{Q}^*) = \bar{Q}^*$. Due to the difference between the definitions of (HQ) and the aggregated quality function Q^*, the learning algorithm adopting this updating rule is not guaranteed to produce the aggregated

qualities given in equation (4.6). **Q.E.D**

Notice that the proof of Proposition 2 does not rely on assumptions A1–A4 made in Section 4.2. This observation shows that the coarse-grained learning algorithm converges even without the four assumptions.

Proposition 3 *By adopting a fixed learning policy π (i.e. the fixed policy learning strategy), the fine-grained learning algorithm converges, provided that the discount rate α is chosen properly*[2] *and the learning rate γ_m (as in equation (4.31)) satisfies:*

$$\sum_{m=0}^{\infty} \gamma_m = \infty, \sum_{m=0}^{\infty} \gamma_m^2 < C < \infty$$

In order to prove Proposition 3, one theoretical result from the stochastic approximation research [65, 227] is used. It is presented as Theorem 5.

Theorem 5 *Suppose that the updating rule of a learning algorithm is of the form*

$$\vec{b} \leftarrow \vec{b} + \gamma_m \cdot \left(A \cdot \vec{b} + D + w\right)$$

where w is a zero mean noise term, and is independent from one learning iteration to another. A is a $N \times N$ matrix. D is a N-vector. Here, N denotes the dimension of vector $\vec{b}$. The learning rate γ_m satisfies

$$\sum_{m=0}^{\infty} \gamma_m = \infty, \sum_{m=0}^{\infty} \gamma_m^2 < C < \infty$$

If the matrix A is negative definite, the algorithm converges to a vector $\vec{b}^*$, which is the solution of the equation

$$A \cdot \vec{b}^* + D = \vec{0}$$

Relying on Theorem 5, the proof of Proposition 3 is as follows.

Proof: Suppose that there are N fuzzy rules within the fuzzy rule base, and $\vec{b}$ is a real vector of N dimensions. Further assume that at most $\mathcal{N}$ domain methods can be performed in order to achieve all the given tasks. At each time when all the tasks conclude, the output vector $\vec{b}$ is updated according to

[2]The conditions satisfied by the discount rate α will be clarified during the proof of this Proposition.

$$\vec{b} \leftarrow \vec{b} + \gamma_t \sum_{i=0}^{\mathcal{N}-1} Z_i \cdot d_{S_i,S_{i+1}}$$
$$= \vec{b} + \gamma_t$$
$$\sum_{i=0}^{\mathcal{N}-1} \left\{ Z_i \cdot \left(\begin{array}{c} Avg \\ \left(\begin{array}{c} S_{i+2}, M_{i+1}, \\ (G_M)_{i+1} \end{array} \right) \end{array} \left(\begin{array}{c} R(S_i, S_{i+1}) + \alpha \cdot \\ \phi(S_{i+1}, S_{i+2}, M_{i+1}, (G_M)_{i+1})^T \cdot \vec{b} \\ -\phi(S_i, S_{i+1}, M_i, (G_M)_i)^T \cdot \vec{b} \end{array} \right) \right) \right\}$$

Obviously, the updating process determined by the above equation is equivalent to the fine-grained updating rule given in equations (4.31) and (4.32). Making this statement, assumption A3 is implicitly utilized, which implies that for each tuple (S, S', M, G_M) with the fixed system state S, the blocks to which S' or M belongs must be different.

Notice that for any tuple (S, S', M, G_M) such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, $G_M \in Is_g$, and $S \xrightarrow{M} S'$, $\phi(S, S', M, G_M)$ is identical. Therefore, the average operation presented in equation (4.36) returns the same vector, denoted as $\tilde{\phi}(S, S', M, G_M)$. $\tilde{\phi}$ is a function that maps any tuple (S, S', M, G_M) satisfying the same condition above to a single vector. As a result, a more compact representation of the updating process is arrived as in equation (4.37).

$$\underset{(S_{i+2}, M_{i+1}, (G_M)_{i+1})}{Avg} \left(\phi(S_{i+1}, S_{i+2}, M_{i+1}, (G_M)_{i+1}) \right) \tag{4.36}$$

$$\vec{b} \leftarrow \vec{b} + \gamma_t \sum_{i=0}^{\mathcal{N}-1} \left\{ Z_i \cdot \left(\begin{array}{c} R(S_i, S_{i+1}) + \alpha \cdot \tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T \cdot \vec{b} - \\ \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T \cdot \vec{b} \end{array} \right) \right\} \tag{4.37}$$

Defining a matrix A and a vector D as

$$A = E \left\{ \sum_{i=0}^{\mathcal{N}-1} Z_i \cdot \left[\alpha \cdot \tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T - \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T \right] \right\}$$

and

$$D = E \left\{ \sum_{i=0}^{\mathcal{N}-1} Z_i \cdot R(S_i, S_{i+1}) \right\}$$

The updating process given in equation (4.37) is simplified as

$$\vec{b} \leftarrow \vec{b} + \gamma_t (A \cdot \vec{b} + D + w) \tag{4.38}$$

where w is a zero mean noise term. By utilizing Theorem 5, if the matrix A is negative definite, then the corresponding fine-grained learning algorithm is convergent. The proposition is consequently proved.

The remaining part of this proof considers the condition under which the matrix A is negative definite. Particularly,

$$
\begin{aligned}
A &= E\left\{\sum_{i=0}^{\mathcal{N}-1} Z_i \cdot \left[\alpha \cdot \tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T - \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T\right]\right\} \\
&= E\left\{\sum_{i=0}^{\mathcal{N}-1}\sum_{m=0}^{i}\left[\begin{array}{c}(\alpha\lambda)^{i-m}\phi(S_m, S_{m+1}, M_m, (G_M)_m) \\ \cdot\left(\begin{array}{c}\alpha\tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T - \\ \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T\end{array}\right)\end{array}\right]\right\} \\
&= E\left\{\sum_{i=0}^{\infty}\sum_{m=0}^{i}\left[\begin{array}{c}(\alpha\lambda)^{i-m}\phi(S_m, S_{m+1}, M_m, (G_M)_m) \\ \cdot\left(\begin{array}{c}\alpha\tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T - \\ \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T\end{array}\right)\end{array}\right]\right\}
\end{aligned}
$$

The last step takes the convention $\phi(S_i, S_{i+1}, M_i, (G_M)_i) = \vec{0}$ for $i \geq \mathcal{N}$.

Let $\mathcal{Q}_m$ be a diagonal matrix with diagonal entries indicated by $q_m(S, S', M, G_M)$. For any tuple (S, S', M, G_M), $q_m(S, S', M, G_M)$ refers to the probability of observing the tuple when the system has undergone state changes m times. $\mathcal{Q}_m$ is therefore a $n^2 \cdot \|\mathcal{M}\|$ dimensional square matrix, where n is the cardinality of the set $\mathcal{S}$. $\|\mathcal{M}\|$ is the cardinality of the set $\mathcal{M}$.

Define a $(n^2 \cdot \|\mathcal{M}\|) \times N$ matrix Φ. For each tuple (S, S', M, G_M), the row vector $\phi(S, S', M, G_M)^T$ becomes one separate row of Φ. Additionally, the $n^2 \cdot \|\mathcal{M}\|$ dimensional square matrix P gives us the transition probabilities among the tuples (S, S', M, G_M). Each entry of P, p_{ij}, indicates the probability of observing the tuple indexed by j when the tuple indexed by i has been observed. With these notations, the following two equations, which have been proved in [29], are obtained.

$$
E\left[\phi(S_m, S_{m+1}, M_m, (G_M)_m) \cdot \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T\right] = \Phi^T \mathcal{Q}_m P^{i-m} \Phi
$$

and

$$
E\left[\phi(S_m, S_{m+1}, M_m, (G_M)_m) \cdot \tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T\right] = \Phi^T \mathcal{Q}_m P^{i-m} \tilde{\Phi}
$$

where the matrix $\tilde{\Phi}$ is similarly defined as the matrix Φ. Each row of $\tilde{\Phi}$ gives the row vector $\tilde{\phi}$ for a specific tuple (S, S', M, G_M). Using these two equations, matrix A is further represented as

$$
\begin{aligned}
A &= \Phi^T \left[\sum_{i=0}^{\infty}\sum_{m=0}^{i} \alpha \cdot \mathcal{Q}_m \cdot (\alpha\lambda P)^{i-m}\right] \tilde{\Phi} \\
&\quad -\Phi^T \left[\sum_{i=0}^{\infty}\sum_{m=0}^{i} \mathcal{Q}_m \cdot (\alpha\lambda P)^{i-m}\right] \Phi \\
&= \Phi^T \cdot \alpha \cdot \mathcal{Q}\left[\sum_{i=0}^{\infty}(\alpha\lambda P)^i\right]\tilde{\Phi} - \Phi^T \cdot \mathcal{Q} \cdot \left[\sum_{i=0}^{\infty}(\alpha\lambda P)^i\right]\Phi
\end{aligned}
$$

where the matrix $\mathcal{Q}$ is defined as

$$\mathcal{Q} = \sum_{i=0}^{\infty} \mathcal{Q}_i$$

Because $\mathcal{Q}_m$ decays exponentially with m (one property of Markov Process), matrix $\mathcal{Q}$ is finite. It has positive diagonal entries and is positive definite. Since $\tilde{\Phi}$ is derived from Φ through certain average operations (refer to equation (4.36)), a relation between $\tilde{\Phi}$ and Φ is established as

$$\tilde{\Phi} = \tilde{P} \cdot \Phi$$

where $\tilde{P}$ is a $n^2 \cdot \|\mathcal{M}\|$ dimensional square matrix. Since matrix P is of the same dimension, there exists one matrix $\tilde{P}$ such that

$$\tilde{P} = P + P_E$$

With this equation,

$$\begin{aligned} A &= \Phi^T \mathcal{Q} \sum_{i=0}^{\infty} \left[\lambda^i (\alpha P)^{i+1}\right] \Phi - \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\lambda P)^i\right] \Phi \\ &\quad + \phi^T \mathcal{Q} \alpha \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \Phi \\ &= \Phi^T \mathcal{Q} \left[(1-\lambda) \sum_{i=0}^{\infty} \left(\lambda^i (\alpha P)^{i+1}\right) - I\right] \Phi \\ &\quad + \alpha \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \\ &= \Phi^T \mathcal{Q} (M - I) \Phi + \alpha \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \Phi \end{aligned} \tag{4.39}$$

where M is defined as

$$M = (1-\lambda) \cdot \sum_{i=0}^{\infty} \left[\lambda^i (\alpha P)^{i+1}\right]$$

As the matrix A is an $N \times N$ matrix, denote the vector produced by the product $A \cdot \vec{b}$ as $\vec{J}$. If $\vec{b}$ is the fuzzy rule output vector, $\vec{J}$ actually becomes the list representation of the function $\Delta Q'$. Now consider the multiplication performed on the first term of equation (4.39).

$$\begin{aligned}\vec{b}^T \cdot [\phi \mathcal{Q}(M-I)\phi] \cdot \vec{b} \\ = \vec{J}^T \mathcal{Q}(M-I)\vec{J}\end{aligned}$$

Define the vector norm $\| \bullet \|_{\mathcal{Q}}$ to be

$$\|\vec{J}\|_{\mathcal{Q}}^2 = \vec{J}^T \mathcal{Q} \vec{J}$$

According to the research presented in [29],

$$\|P\vec{J}\|_{\mathcal{Q}} \leq \|\vec{J}\|_{\mathcal{Q}}$$

and

$$\|P^m \vec{J}\|_{\mathcal{Q}} \leq \|\vec{J}\|_{\mathcal{Q}}, \forall \vec{J}, m \geq 0$$

Subsequently,

$$\begin{aligned}\|M\vec{J}\|_{\mathcal{Q}} &\leq (1-\lambda)\sum_{i=0}^{\infty}\left[\lambda^i \alpha^{i+1}\|\vec{J}\|_{\mathcal{Q}}\right] = \frac{\alpha(1-\lambda)}{1-\alpha\lambda}\|\vec{J}\|_{\mathcal{Q}} \\ &\leq \alpha\|\vec{J}\|_{\mathcal{Q}}\end{aligned}$$

With the above inequality,

$$\begin{aligned}\vec{J}^T\mathcal{Q}(M-I)\vec{J} &= \vec{J}^T\mathcal{Q}M\vec{J} - \vec{J}^T\mathcal{Q}\vec{J} \\ &= \vec{J}^T\mathcal{Q}^{1/2}\mathcal{Q}^{1/2}M\vec{J} - \|\vec{J}\|_{\mathcal{Q}}^2 \\ &\leq \|\mathcal{Q}^{1/2}\vec{J}\|_2 \cdot \|\mathcal{Q}^{1/2}M\vec{J}\|_2 - \|\vec{J}\|_{\mathcal{Q}}^2 \\ &= \|\vec{J}\|_{\mathcal{Q}} \cdot \|M\vec{J}\|_{\mathcal{Q}} - \|\vec{J}\|_{\mathcal{Q}}^2 \\ &\leq \alpha\|\vec{J}\|_{\mathcal{Q}}^2 - \|\vec{J}\|_{\mathcal{Q}}^2 \\ &= -(1-\alpha)\|\vec{J}\|_{\mathcal{Q}}^2\end{aligned}$$

$\| \bullet \|_2$ is the Euclidean vector norm. Suppose that the discount rate $\alpha \leq \alpha^c$, where $0 < \alpha^c < 1$, then for any vector $\vec{J} \neq \vec{0}$,

$$\vec{J}^T\mathcal{Q}(M-I)\vec{J} \leq -(1-\alpha^c)\|\vec{J}\|_{\mathcal{Q}}^2 < 0$$

Consequently, for any N dimensional vector $\vec{b}$, $\vec{b} \neq \vec{0}$,

$$\vec{b}^T A\vec{b} \leq (\alpha^c - 1)\|\vec{J}\|_{\mathcal{Q}}^2 + \alpha\Lambda(\vec{J})$$

where $\vec{J} = \Phi \cdot \vec{b}$, function $\Lambda(\vec{J})$ is defined as

$$\Lambda(\vec{J}) = \vec{J}^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \vec{J}$$

As P^i decreases to 0 exponentially with i, function $\Lambda(\vec{J})$ is continuous. For any vector $\vec{b}$ with bounded vector norm, $\Lambda(\Phi\vec{b})$ is bounded as well. Suppose that the output vector $\vec{b}$ adjusted by the fine-grained learning algorithm is bounded within a region Re (this assumption can be ensured by adopting the trapping function introduced in Section 4.5), there exists correspondingly a discount rate $\alpha \leq \alpha^c$ such that

$$\vec{b}A\vec{b} \leq (\alpha^c - 1)\|\Phi\vec{b}\|_{\mathcal{Q}}^2 + \alpha \cdot \Lambda(\Phi\vec{b}) < 0$$

From this inequality, it is concluded that by properly choosing the discount rate α, the matrix A is negative definite for all vectors $\vec{b}$ within the region Re. Consequently, the fine-grained learning algorithm with fixed learning policy (i.e. the fixed policy learning strategy) is convergent. This ends the proof of Proposition 3. **Q.E.D**

In the last part of this Section, the fourth theoretical result regarding the fine-grained learning algorithm is presented.

Proposition 4 *The fine-grained learning algorithm, which adopts an updating process governed by equation (4.34), converges, provided that Proposition 3 is valid, the step-weight δ is properly selected, and the learning rate γ_t of equation (4.34) satisfies the following conditions:*

$$\sum_{t=0}^{\infty} \gamma_t = \infty, \sum_{t=0}^{\infty} \gamma_t^2 < C < \infty$$

Proof: To simplify the discussion, only the case where the output vector $\vec{b}$ to be updated by equation (4.31) is within a bounded region around $\vec{0}$ will be considered. Actually, due to the trapping function $trap(\bullet)$, the vector $\vec{b}$, which is produced by equation (4.31) and is outside the bounded region, is re-mapped into the region. Thus, equation (4.34) can be re-written as

$${}^2\vec{b} \leftarrow (1 - \gamma_t) \cdot {}^2\vec{b} + \gamma_t \cdot H({}^2\vec{b})$$

By utilizing traditional stochastic approximation theories (i.e. *Proposition 4.4* in [29]), if $H(\bullet)$ is a contraction mapping, the corresponding learning algorithm is then convergent. The remaining part of this proof therefore examines the circumstances under which the mapping $H(\bullet)$ are contractive. Notice that for differed fuzzy rule output vectors $\vec{b}_1$ and $\vec{b}_2$, the corresponding learning policies are different. This difference leads to varied matrix A and vector D of equation (4.38). Denote the specific matrix A determined by the vector $\vec{b}$ as $A_{\vec{b}}$, and $D_{\vec{b}}$ for vector D obtained by setting the fuzzy rule output vector as $\vec{b}$. Additionally, since the vectors $\vec{b}$ to be updated by the fine-grained learning algorithm are bounded within a pre-defined region (due to the trapping function), Re is used to denote the set of all possible vectors $\vec{b}$ that can be processed by the algorithm. Define further $E_A(\vec{b}_1, \vec{b}_2)$ and $E_D(\vec{b}_1, \vec{b}_2)$ respectively as

$$E_A = |||A_{\vec{b}_1} - A_{\vec{b}_2}|||$$

and

$$E_D = \|D_{\vec{b}_1} - D_{\vec{b}_2}\|$$

where $||| \bullet |||$ is any matrix norm. In order to show that $H(\bullet)$ is a contractive mapping, for any two vectors $\vec{b}_1$ and $\vec{b}_2$ of Re, the following inequality must be valid:

$$\|H(\vec{b}_1) - H(\vec{b}_2)\| < \|\vec{b}_1 - \vec{b}_2\| \tag{4.40}$$

According to Theorem 5, suppose that the vectors ${}^2\vec{b}_1$ and ${}^2\vec{b}_2$ satisfy the conditions:

$$A_{\vec{b}_1} \cdot^2 \vec{b}_1 + D_{\vec{b}_1} = \vec{0},$$
$$A_{\vec{b}_2} \cdot^2 \vec{b}_2 + D_{\vec{b}_2} = \vec{0}$$

Inequality (4.40) can be re-written as

$$\delta \cdot \|^2\vec{b}_1 -^2 \vec{b}_2\| < \|\vec{b}_1 - \vec{b}_2\| \tag{4.41}$$

Consequently, to show that $H(\bullet)$ is contractive is to show that inequality (4.41) is valid for all $\vec{b}_1, \vec{b}_2 \in Re$. Derived from the error bounds on solutions of linear systems [141],

$$\|^2\vec{b}_1 -^2 \vec{b}_2\| \leq X \cdot E_A(\vec{b}_1, \vec{b}_2) + Y \cdot E_D(\vec{b}_1, \vec{b}_2)$$

where X and Y are two positive numbers. Since both functions E_A and E_D are continuous and bounded for all $\vec{b}_1, \vec{b}_2 \in Re$, there exists a step-weight δ such that inequality (4.41) is valid. Thus, by choosing a proper δ, the fine-grained learning algorithm under the restricted policy iterations is convergent. **Q.E.D**

The proof of Proposition 4 suggests that during each learning phase, the updating process of determining the vector ${}^1\vec{b}$ need not to converge. Several updating iterations might suffice. After that, the algorithm proceeds and another learning phase begins. This is due to the fact that the updating rule given in equation (4.34) permits an additional noise term w, provided that $H(\bullet)$ is a contractive mapping.

Complementing to this theoretical analysis, which shows that the two learning algorithms are convergent, the experimental studies of the learning algorithms is presented in the next section.

4.7 Experimental Evaluation: A multi-agent pathology lab system

The multi-agent pathology lab system introduced in Section 3.5 is used to evaluate the effectiveness of the two learning algorithms. In the implementation, 5 randomly selected samples will be supplied to the lab at each time when the system starts. A distinct task is created for every sample. The overall system performance is measured according to Equation (3.5) and (3.6). Specifically, Table 4.1 summarizes the actual values of those constants used in the evaluation of rewards and the system performance. As shown in Table 4.1, each task failure results in a heavy penalty, highlighting the importance of managing hard system constraints.

Str	$rDis$	$pDis$	$failPenalty$	α^c
1	2.1	2.3	-20	0.95

Table 4.1: Constants used in the evaluation of rewards and the system performance.

To investigate the effectiveness of the two learning algorithms, several experiments have been performed. The first experiment is designed around the coarse-grained learning algorithm for managing hard system constraints. One agent uses the coarse-grained learning algorithm to estimate the actual impact of hard constraints on the qualities of performing various methods. In order to decrease the overall computational complexity, D_s (see Section 4.4) in the implementation contains only one fluent which gives the current number of active tasks (i.e., tasks haven't concluded). Accordingly, D'_s includes a single fluent which counts the number of active tasks after the execution of the method. A_f also

involves only one fluent. It indicates the satisfaction degree of the method with respect to the hard relations of the corresponding task. Two criteria are used to measure the performance of our learning algorithm, the *failure probability* and the *total reward* obtained. The *failure probability* refers to the probability of task failures during each run of the simulated system, and the total reward is evaluate by simple summations of the rewards incurred. The learning results are shown in Table 4.2. They are obtained by averaging 500 independent evaluations of the learning algorithm.

Learning Times	500	1000	2000	3000	4000	5000
Failure Prob.	0.35	0.154	0.104	0.116	0.076	0.054
Avg. Performance	-36.3	-18.0	-13.3	-14.6	-11.1	-9.06

Table 4.2: The performance of the coarse-grained learning algorithm by one learning agent.

The experiment shows that the learning result converges after 5000 times of learning (Ref. Table 4.2). No more improvement can be achieved afterwards. In addition, the failure probability did not decrease to 0. This is because certain hard relations cannot be strictly satisfied. Notice that this performance result can be further improved if the coarse-grained learning algorithm exploits extra fluents, as the third experiment to be described shortly shows.

The second experiment employs two learning agents. The fluents utilized are identical to the previous experiment. The learning results are stored in the same table and updated simultaneously by the two agents. The data obtained from the experiment is shown in Table 4.3.

Learning Times	500	1000	2000	3000	4000	5000
Failure Prob.	0.362	0.206	0.170	0.072	0.056	0.054
Avg. Performance	-34.8	-20.0	-16.3	-5.41	-3.84	-3.61

Table 4.3: The performance of the coarse-grained learning algorithm for the situation of two learning agents.

Compared with data in Table 4.2, Table 4.3 indicates that the convergence rate of the coarse-grained learning algorithm is around 30% faster than the case of a single learning agent. In addition, due to the concurrent operations of the two agents, the average performance of the learned polices is about 1.8 times higher than the one in the first experiment.

In the third experiment, system performance is further improved by introducing more fluents into the coarse-grained learning algorithm. Specifically, D_s is extended with one additional fluent which gives the information about the ex-

pected processing time that remains in order to conclude all the active tasks. Accordingly, a new fluent is included in D'_s, which provides the expected total processing time after the completion of the corresponding method. The experiment is carried out with two learning agents. The obtained results are summarized in Table 4.4.

Learning Times	500	1000	2000	3000	4000	5000
Failure Prob.	0.384	0.252	0.204	0.076	0.062	0.054
Avg. Performance	-37.0	-25.3	-21.2	-6.01	-3.62	-2.51

Table 4.4: The performance of the coarse-grained learning algorithm with an extended fluent set and two learning agents.

Table 4.4 highlights that the system performance can be indeed further improved by introducing extra fluents into the coarse-grained learning algorithm. However, as our third experiment shows, the difference is sometimes not salient enough as compared with the increased computational complexity. The data given in Table 4.2, 4.3, and 4.4 are compared in Figure 4.9.

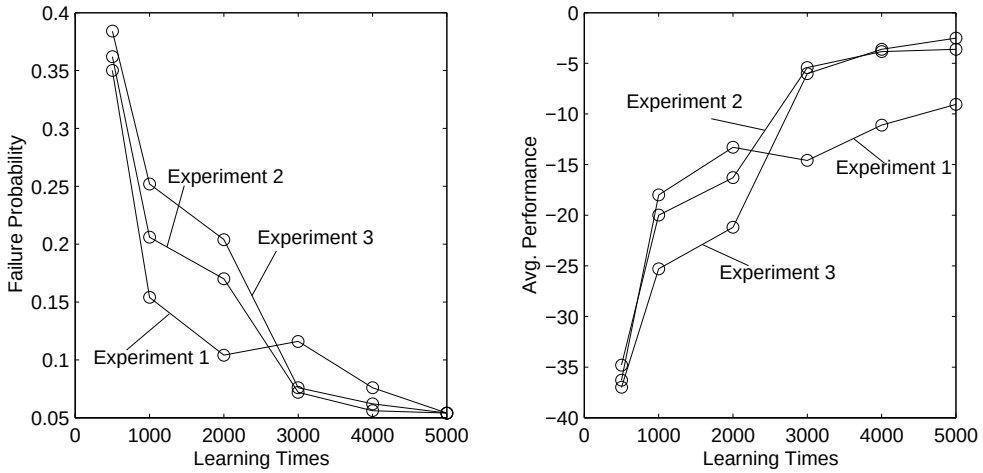

Figure 4.9: Performance of the coarse-grained learning algorithm.

To investigate the scalability of the algorithm in terms of the number of learning agents, similar experiments for 3 to 8 agents have been performed. A total of 3 fluents are utilized by the learning algorithm. The learning procedure generally converges more quickly than the cases of 1 and 2 learning agents. The system performance achieved after the learning procedure converges is summarized in Table 4.5, and the changes of performance with respect to the number of learning agents is shown in Figure 4.10.

Table 4.5 and Figure 4.10 show that by employing more learning agents, the system performance is further improved. Nevertheless, no extra performance improvement can be observed after 5 learning agents are used. This is not surprising

Number of Agents	3	4	5	6	7	8
Failure Prob.	0.056	0.055	0.056	0.054	0.055	0.56
Avg. Performance	-2.75	-2.61	-2.52	-2.49	-2.51	-2.53

Table 4.5: The performance of the coarse-grained learning algorithm with respect to the number of learning agents.

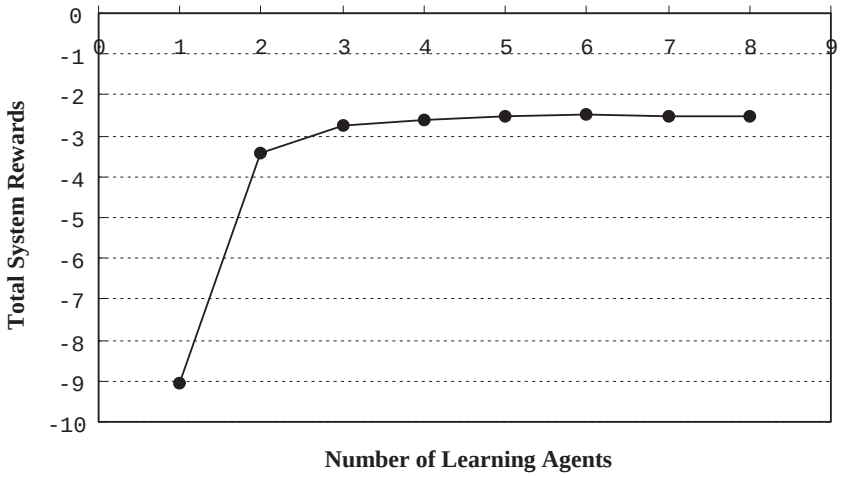

Figure 4.10: The changes of performance with respect to the number of learning agents.

as within the simulated pathology lab, at most 5 pathological samples can be processed concurrently. As a consequence, any system performance improvement cannot be observed since, at any time, only 5 agents can contribute effectively to the production of pathological specimens.

Besides showing that the learning algorithms are effective, the algorithms also appear to be good choices for task-oriented domains. Specifically, the learning results are compared with a genetic algorithms-based learning scheme similar to the one proposed in [34]. Instead of using reinforcement learning methods, this approach uses the genetic algorithms to adjust the fuzzy rules. Within the fuzzy rule base, 40 fuzzy rules are constructed to estimate the aggregated quality function in the presence of only hard system constraints. The preconditions of these rules are taken directly from the fuzzy rules established through the learning results of our first experiment. The outputs of these rules form a vector, which is treated as an individual chromosome of the genetic algorithm. In the implementation of the scheme, 20 chromosomes form a generation. In order to carry out one generation iteration, each of the 20 chromosomes must be evaluated independently. For any chromosome to be evaluated, each fuzzy rule output is first adjusted according to the chromosome. The derived fuzzy rule base is consequently employed by each

agent (2 agents are deployed in the experiment) of the simulated pathology lab system. The system performance observed with this configuration is then treated as the fitness value of that chromosome. In order to ensure that the fitness value does not deviate too much under repeated evaluations of the same chromosome, an average of 50 independently observed system performances is used as the evaluated fitness. Given this brief description of the *GA-based scheme*, Figure 4.11 shows the average system performance achieved with respect to 50 consecutive generation iterations.

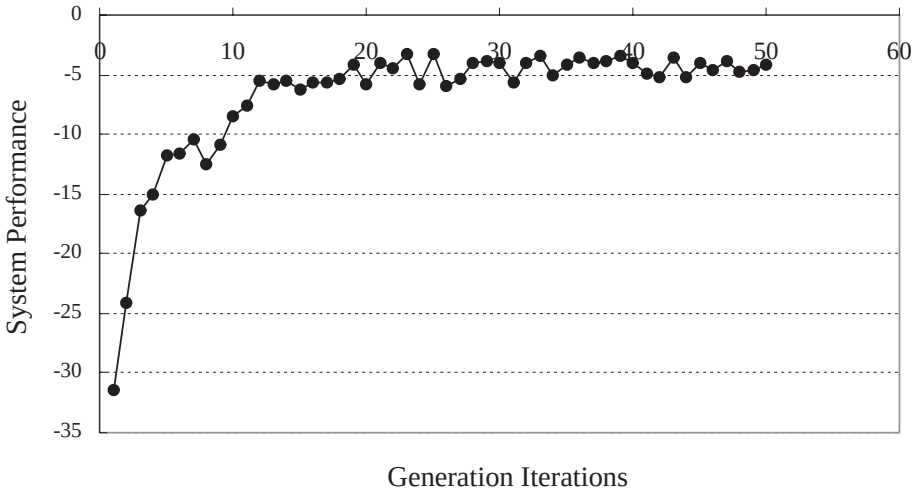

Figure 4.11: The average system performance achieved for 50 consecutive generation iterations.

As shown in Figure 4.11, it takes around 20 generation iterations before the system performance reaches -5. After that, the performance vibrates constantly around -3.8. The best performance achieved is -3.456 at the 25th generation. This performance is slightly worse than that achieved by the coarse-grained learning algorithm.

Compared with the experiments described previously, at least two major advantages of the learning algorithms can be identified:

1. Since the fitness value evaluated for each chromosome is not deterministic, there is no guarantee that the GA based scheme converges. In contrast, the learning algorithms can offer certain convergence guarantees as the theoretical analysis (Ref. Section 4.6) has already shown.

2. The computational complexity of the genetic algorithm tends to be significantly higher than that of the learning algorithms. For instance, to reach a system performance of -5, the genetic algorithm requires a total of

$20 \times 50 \times 20 = 20000$ independent runs of our simulated pathology lab system. Conversely, to achieve a similar system performance, the coarse-grained learning algorithm requires only 4000 independent simulations. These results suggest that the learning algorithms can be 5 times faster than the GA-based scheme.

This provides a further argument that the algorithms proposed in this Chapter can be efficient learning approaches for agent coordination in task-oriented environments.

Besides investigating the coarse-grained learning algorithm, the last two experiments are performed to evaluate the fine-grained learning algorithm. The fine-grained algorithm is designed to be used in conjunction with the coarse-grained learning algorithm to adjust the output parts of 4 previously given fuzzy rules. Each rule gives an estimation of the expected quality changes, taking into account of the soft relation (Ref. Figure 3.4). The learning results obtained from the second experiment are utilized to facilitate the fine-grained learning process. Figure 4.12 depicts the overall changing process of fuzzy rule outputs when λ is set to 0.3. The experiment is carried out with two learning agents under the fixed policy learning strategy (Ref. Section 4.5).

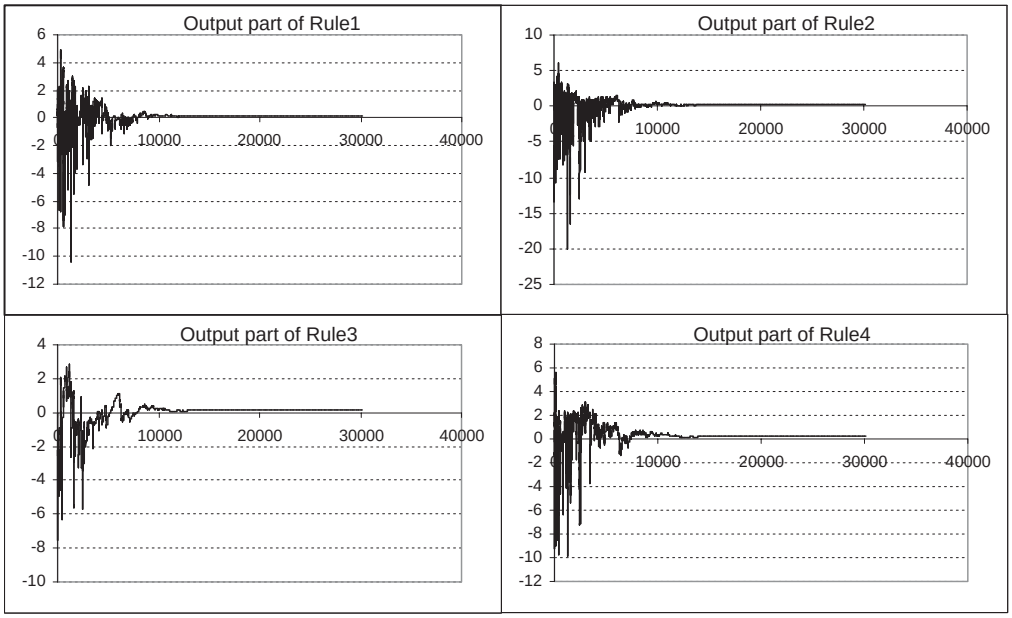

Figure 4.12: The changing process of the 4 fuzzy rule outputs. The fine-grained learning algorithm takes the fixed policy learning strategy. The horizontal axis denotes the times of learning.

Figure 4.12 shows that the learning algorithm converges after about 10000 times of learning. The convergence rate slightly changes with the variation of λ. Basically, a value around 0.5 for λ in the simulated system contributes to

a faster convergence rate and a higher system performance. Table 4.6 gives the experimental data obtained for λ equals to 0.3, 0.5, and 0.9 respectively. Although the system performance for these three settings is different, it is evidenced that the fine-grained learning algorithm is effective in managing soft system constraints.

λ	Failure Probability	Average Performance Improvement
0.3	0.054	1.58
0.5	0.0538	2.31
0.9	0.054	1.73

Table 4.6: The performance of the fine-grained learning algorithm.

Figure 4.13 shows the performance of the fine-grained learning algorithm under the restricted policy iterations (Ref. Section 4.5). As implemented in this experiment, during each learning phase, the updating procedure determined by equation (4.31) is carried out only once. As indicated in Section 4.6, this treatment would not prevent the algorithm from converging, assuming that $H(\bullet)$ is a contractive mapping. The specific configuration is as follows: ν and κ of the function $trap^*(\bullet)$ are set to 10 and 15, respectively; the step-weight δ equals to 0.5; and λ takes value 0.5.

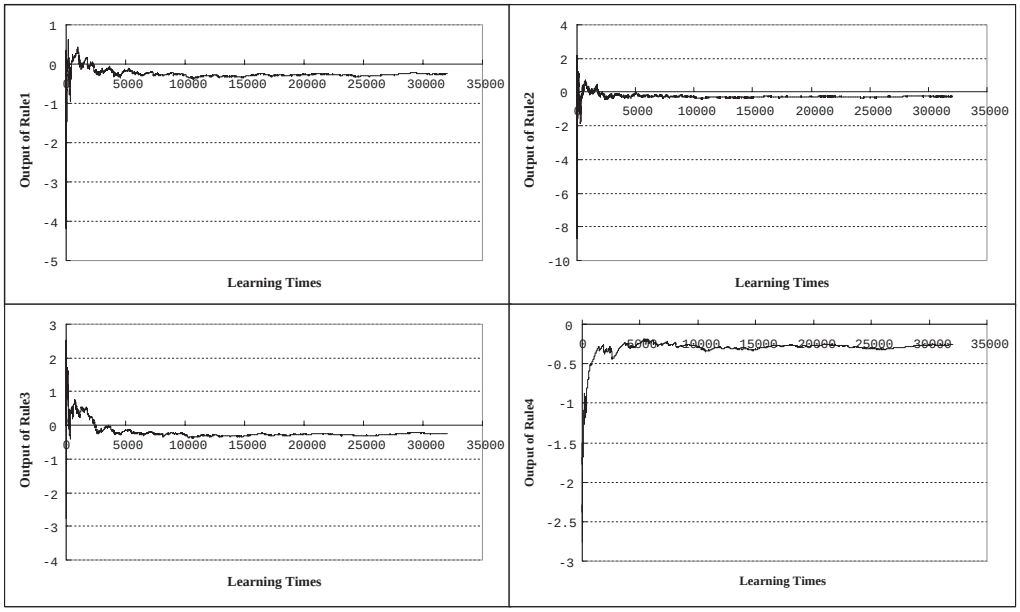

Figure 4.13: The changing process of the 4 fuzzy rule outputs. The fine-grained learning algorithm takes the *restricted policy iteration* learning strategy.

As compared with Figure 4.12, Figure 4.13 shows a much smoother changing process of each fuzzy rule output. This suggests that the corresponding system performances do not undergo drastic changes after incorporating these fuzzy rules into agents' decision-making procedures. Besides, the algorithm converges after

about 5000 times of learning. This result shows that the restricted policy iteration is more time efficient than the fixed policy learning strategy. Nevertheless, contrary to the expectation when introducing the restricted policy iteration, the improvement of the system performance after the learning process ends is around 1.94. It is lower than that achieved by the fixed policy learning strategy when λ equals to 0.5. It seems that the introduction of the step-weight δ, which changes artificially the actual fuzzy rule outputs learned through equation (4.31), should take the blame. However, for some applications, the step-weight δ is necessary to ensure that the algorithm is convergent. These results indicate that the two learning strategies of the fine-grained learning algorithm are equally important, and they will find their practical use in various application domains.

4.8 Related Work

The agent coordination issues have been originally addressed in the Distributed Artificial Intelligence (DAI) research [35]. Coordination among agents is usually achieved through distributed planning and scheduling techniques [91, 103, 244]. Recently, agent coordination via reinforcement learning has received much research interests [56, 155, 260, 266, 286]. It provides a new way of designing agents in which agents' interaction experiences are explored without specifying *a priori* how the requirements are to be satisfied [29]. The success in agent-based reinforcement learning also suggests that learning methods may provide effective, near-optimal solutions to complex planning and decision-making problems under uncertain environments [83,256]. For example, the Q-learning algorithm has been exploited to coordinate multiple agents controlling elevators [83, 285]. The learning results are stored in a neural network, and the Q-learning algorithm constantly adjusts network's weights. Different from this approach, the research reported in this Chapter utilizes state aggregation techniques, and the learning algorithms are designed to estimate an aggregated quality function, which is different from typical Q-learning algorithms. Besides, the learning results are provided as a set of fuzzy rules. Adopting the fuzzy rules, agents are capable of handling a variety of system fluents, which take either numerical or symbolic values. The adoption of aggregation techniques enable the two learning algorithms to operate with a potentially huge state space. Compared with using neural networks, the learned rules are also more intuitive to human designers.

Learning based approach, particularly via reinforcement learning, to coordinating agents was reported in [3, 7, 25, 249]. Similar to ours, much of these works is centered around the issue of job and resource allocations among multiple agents [25,286]. Nevertheless, they differ in both the applied problem domains and

the interaction mechanisms explored. For example, the ACE and AGE algorithms proposed by Weiss have adopted the bidding mechanism to facilitate the overall learning process [286]. On the other hand, in the work of Abul *et al.*, the effects of other agents' behaviors can only be acquired through self-observations [3]. Although their results are promising, most of works take the information of agents' dependencies implicitly. Where approximation techniques were used, some approximation structures as adopted (e.g. fuzzy rules) may fail to utilize such important information as agents' dependencies, and thus the overall system performance suffers. This is the case in [25], which has applied reinforcement learning methods in a grid-world environment. Without considering the relations among the actions of different agents, the decision policy derived directly from the fuzzy rules as given in [25] may lead to uncoordinated behaviors among multiple agents. For example, the agents may concurrently pursue one action, and thus waste their resources and efforts. The downgrading of the overall system performance results.

The importance of exploring coordination-specific information (e.g. the inter-dependencies among agents) in learning algorithms was noticed in the literature, such as [277]. However, no systematic approach to utilizing that kind of information was proposed. The focus was placed only on specific application domains. Additionally, theoretical analysis of their learning algorithms was not provided.

The two reinforcement learning algorithms proposed in this chapter relies on the FSTS model (see Chapter 3) that models agent coordination as a Decision Theoretical Planning problem (DTP) [36]. DTP in turn roots in the Markov Decision Process (MDP). In recent years, Markov based approach has become a hot topic to model various kinds of agent interations. Besides reinforcement learning, the literature abounds and presents a variety of system assumptions and solution methods [18, 20, 156, 224, 284, 294]. Several research works that follow the Markov based approach will be introduced in the sequel in order to give a high-level picture of the recent research progress in this field.

In [284], coordinating actions with a single agent is considered. The agent's behavior is modeled as a Partially Observable Markov Decision Process (POMDP). The research intends to identify good responses to the actions of the agent. The scope and focus is different from the two reinforcement learning algorithms proposed in this chapter. In this chapter, the task-oriented application domain instead of an agent is model as a MDP where multiple agents cooperate with each other in order to finish a given group of tasks. The learning objective is to enhance the inter-agent cooperation by recognizing and utilizing both hard and soft relations. No concepts such as the "*best responses*" are involved in the research.

The research reported in [224] takes a similar methodology as in [284]. In [224], Rabinovich and Rosenschein extends the applicability of Markov tracking

to cater the situation of more than 2 agents. Each agent estimates the exhibited system dynamics based on its own observational data, choose the joint action, and perform its part in it. To identify a well-fitted joint action, dynamic programming methods are used to solve the MDP recognized. Differing from this research, the two reinforcement learning methods in this chapter seeks to identify joint actions without estimating a MDP in the first place. Meanwhile, structures in tasks are utilized to aggregate system states and domain methods. Through this technique, the two learning algorithms are easily extendable to handle large problem spaces. In comparison, performance issues for dynamic programming are omitted in [224].

In [20], Becker *et. al.* summarize a series of research work on solving a particular class of Decentralized Markov Decision Processes (DEC-MDP). The research is carried out by Prof. Lesser's group at University of Massachusetts Amherst. The initial result was published in [19] and the application domain was to control multiple planetary exploration rovers. They assume that the global system state is factorable in which the agents' transitions are independent. The class consists of independent collaborating agents that are tied together through a structured global reward function. Agents' decision policies are determined offline by using a so-called Coverage Set Algorithm (CSA), which means that these agents cannot constantly improve their performance during run-time. Focusing on a wider scope of applications that can be categorized as task-oriented domains, the two learning algorithms proposed in this chapter handles the situations where agents' local states are tightly coupled and one agent can affect other agents' actions by changing its local state. As clearly stated in [20], except specific types of constraints, general soft and hard relations considered in this chapter may not be covered properly by their model. Another distinct difference lies in the problem solving methodologies. Becker *et. al.* assume that the system dynamics are known a priori and the agents' policies are fixed before the system starts operation. In comparison, this chapter assumes that the system dynamics are unknown to agents, who can instead use fine-grained learning algorithms during run-time to constantly improve their coordination decisions.

In another research paper [18], Becker *et. al.* apply the same problem solving methodology proposed in [19, 20] to a different domain. Rather than restricting agents' local states to be independent, Becker *et. al.* considers the case when agents' actions can interact with each other through enable and facilitate relations. In order to achieve this, the global reward function is simplified to become a linear summation of every agent's local reward. With this assumption, agents are decoupled from the global reward function, which is an essential character in [20]. As a major difference, the reward function considered in chapters 3 and 4 is to evaluate the task completing performance, which is defined globally without

necessarily involving agents' local rewards.

Other than taking actions, agents can also interact with each other through communication. In [294], Xuan *et. al.* seek to model communication activities within a MDP framework. In addition to decide which action to perform, agents have to determine when they should exchange information of their local states. The research focused on evaluating the impact of communication to agents' local decision-making. Our research, however, provides a communication protocol (see Chapter 5) to ensure the availability of the latest global knowledge. Xuan *et. al.* also take the assumption that the global system state is a combination of agents' local states. This chapter does not rely on such assumptions.

MDP and reinforcement learning alone may not be capable of handling all types of agent interactions. In [142, 156], game theory is explored to solve agent interaction problems. The research objective is to find equilibrium policies between learning agents. In order to achieve this, a quality function that combines every agent's local activities is used by the learning method. However, due to the curse of dimensionality, the learning problem can easily become highly complicated as the number of agents increases. In order to manage the learning cost incurred during exploration, a bayesian model is used in [56] for optimal exploration in a Multiagent MDP.

4.8.1 Equivalent Approaches

In discussing the related work, it often helps if some statements can be made on a possible *equivalence* between the related work and the one presented in this Chapter. But the different assumptions (sometimes drastically different) and different models tend to make it difficult to present a pertinent discussion on the "equivalence" issue. Nevertheless, an attempt is made in this Subsection to present one perspective on the "equivalent approaches".

Under different contexts, the concept of "*equivalence*" can have varied explanations. For example, consider a DTP problem. If it can be shown that the decision policies derived from the two corresponding aggregated systems are actually the same, the two different partitions of the same system are said to be *equivalent* [125]. Within the context of this Chapter, however, because most of learning algorithms execute basically similar iterative updating procedures, two learning based coordination approaches are considered *equivalent* if the system models adopted or domain information utilized by the learning algorithms are identical. With this perspective in mind, two related work reported in the literature will be discussed and shown to be equivalent to the one reported in this Chapter [130, 277].

Vengerov *et al* proposed an adaptive coordination approach to distributed

dynamic load balancing among fuzzy reinforcement learning agents [277]. Agents are designed to work in a *distributed dynamic web caching* environment which is presented as a grid world. The environment is a task-oriented domain in our context. The overall task is to satisfy all the demands for information located at every grid. Each method in the task takes the responsibility of satisfying the exact demand located at a particular grid. Since such demand can only be satisfied by moving an agent to that grid, the notion of meta-methods is not required. There exists one hard system constraint, that is, no more than one agents can be located at the same node at the same time. In the term of FSTS model, the constraint states that no more than one agent can perform concurrently a same method. There are also soft system constraints. Specifically, when multiple agents are satisfying the demands located at a small area, the rewards they can get will be reduced. In other words, the methods can affect each other in terms of rewards obtained. In the view of FSTS, the fine-grained learning algorithm can achieve the agent coordination in the dynamic web caching application *in an equivalent way*. Essentially, the two fluents utilized by the learned fuzzy rules in [277] characterize the state blocks B^S and the soft relations Is_g, respectively. The aggregated quality function they are trying to estimate is consequently of the form $Q^*(B^S, Is_g)$. It can be seen as a specific instance of a more general type of quality functions, $Q^*(B^S, B^{S'}, B^M, Is_g)$. From this perspective, it is reasonable to say that the approach as described in [277] serves as an application example of the learning approaches described in this Chapter.

Notice that the concept of the method relations was not clearly identified in [277]. This shortage was amended in [130]. Guestrin *et al* considered the problem of context specific multiagent coordination and planning with Factored MDPs [130]. In their approach, the relations among various methods to be performed by each agent are represented through *coordination graphs*. A coordination graph for a set of agents is a directed graph. Its nodes, V, correspond to the agents within the system. There is an edge $v_1 \rightarrow v_2$, $v_1, v_2 \in V$, if the methods to be performed by one agent (v_2) are relevant to determining the quality of those methods to be performed by another agent (v_1). It is interesting to note that in contrast to FSTS, this formalism lacks generality in at least two aspects: (1) The coordination graph can only represent the relations among those methods to be performed at present. The affects of previously performed methods cannot be modeled. The graph is unable to represent hard system constraints, which establish actually a specific work-flow among the various domain methods. (2) In the coordination graph, each node corresponds to an individual agent. This formalism is inflexible in a sense that it stipulates that two methods are related if they are performed respectively by two agents who are connected in the graph

(e.g. $v_1 \rightarrow v_2$). Generally, as is the case in FSTS, the relations among agents are determined by the methods they are performing or have performed.

To be fair, given the exact system transition models, the agent coordination problem was nicely dealt with using linear programming techniques in [130]. In this aspect, it is hard to compare their approaches with the two learning algorithms in this Chapter.

4.9 Summary

In this Chapter, reinforcement learning methods have been explored for coordinating multiple agents in a task-oriented environment. Using the FSTS model, the agent coordination can be boiled down to a decision problem and modeled as a Markov Decision Process (MDP). It enables a DTP approach for agent coordination, where reinforcement learning methods can be appropriately applied.

Two learning algorithms, "*coarse-grained*" and "*fine-grained*", were proposed to address agents' coordination behavior at two different levels. The "*coarse-grained*" algorithm operates at one level and tackles *hard* system constraints, and the "*fine-grained*" at another level and for soft constraints. This is a separation of concern, as different system constraints impact the satisfaction of system goals at different level and require different way to address. The experiments demonstrated that the two learning algorithms are effective in coordinating multiple agents, and thus in improving the overall system performance. Additionally, as compared with one genetic algorithm-based learning scheme, the two algorithms also appear to be good learning approaches for task-oriented environments.

Chapter 5

A Protocol to Maintain System State Information in a Multi-agent Environment for Effective Learning

5.1 Introduction

In Chapter 4, agent coordination is viewed as a learning problem where reinforcement learning methods (i.e. the coarse-grained and fine-grained learning algorithms) are applicable. Traditionally in a single agent setting, the problem is typically modeled as a Markov Decision Process (MDP). The overall goal of an agent is to learn a policy in order to maximize the long-term performance (e.g. the accumulation of discounted rewards). In a multi-agent setting such as a task-oriented domain, the MDPs are extended to multi-agent MDPs (MAMDPs) to allow concurrent learning of multiple agents [174]. A MAMDP of m agents is defined as a tuple of the form $M = (S, A^m, \delta, \gamma^m)$, where S is a set of potential system states, A^m is a set of possible actions[1] executable by the m agents, δ is a system transition function, and γ is a set of reward functions for the m agents.

As described in Chapter 3, a system state is considered as a *signal* from the environment [260] where the agents reside in and interact with through performing actions. A system state should include those information that satisfies the *Markov property*. The Markov property states that the next state of a system is determined solely by the current state of the system and actions taken by agents in this state, that is, $\delta : S \times A^m \rightarrow S$. Since the behavior of a MAS system is determined by the whole group of autonomous agents who act independently and concurrently,

• Part of the research results shown in this chapter appears in the *IEICE Transactions on Information and Systems*, 2005.

[1]Following the terminology of the FSTS model (see Chapter 3), the actions are domain methods to be performed by agents. In this Chapter, to comply with the widely adopted terminology of MDPs, the term *action* will be used instead.

the current system state and an agent's discounted reward are not only dependent upon the agent's own policy, but also affected by policies of other agents. Clearly, in a MAS setting, an agent may only observe the partial system state, and thus the introduction of the MAMDP (and Markov property) poses one severe challenge as to how an agent obtains and maintains the current system state.

The case where an agent only observes the partial system state is modeled as the Partially Observable Markov Decision Process (POMDP) [108]. A POMDP problem is hard to solve even in the situation of a single learning agent. Within a multiagent setting, it was often approached under stringent assumptions [174], but no general theoretical results were reported in this regards. In this chapter, a new approach is proposed to help agents maintain a global view which is consistent with the most updated system state [63]. This approach attempts to ensure the *Markov property* of a MAS system. A token-ring-based distributed protocol is presented which satisfies three properties. These properties are theoretically analyzed to guarantee a global system view that respects the Markov property.

The two reinforcement learning algorithms presented in Chapter 4 assume the observation of the full system state and agents utilize the token-ring-based protocol to maintain a consistent view of the global system. The effectiveness of the two algorithms demonstrates the significance of the research reported in this Chapter. In the literature, it has been shown that the algorithms that assume the fully observable system states are both easily understandable and theoretically guaranteed to be effective [142, 143]. While being devised for providing an effective multi-agent learning environment, the protocols are expected to find their applications in other context where a most updated system-level information of a distributed system is required.

The organization of this Chapter is as follows. Section 5.2 models the behaviors and interactions of agents in a multi-agent system. Section 5.3 proposes all the desirable properties of any protocols designed to maintain a global system view. Built on top of these properties, a token-ring-based protocol is presented in Section 5.4. The effectiveness of the protocol is evaluated in Section 5.5 through experiments in the context of two learning algorithms. Finally, Section 5.6 summarizes this Chapter.

5.2 Multi-agent System Model

In a multiagent system, an agent fulfills its missions by executing actions. When an agent starts or finishes the execution of an action, a corresponding event e is said to be fired. An event e occurs at time t is denoted by a tuple (e, t). Assume that there exists a synchronized global time clock with an adequate resolution

for timestampping events. The history at time t, denoted as h^t, is defined as $\{(e, t') \,|t' \leq t\}$. The firing of an event is considered as the result of the decisions made by an agent. It is highly desirable that the agents make their decisions based on the information with the desired *Markov property*. In other words,

R1 *whenever an event fires, the corresponding agent must have the most updated information of the system state.*

A multiagent system π at time t is defined as a tuple:

$$\pi^t = \left(s^t, h^t, \alpha^t\right), s^t \in S, h^t \in H, \alpha^t \subseteq Agent$$

where S and H refer to a set of all possible system states and a set of all possible system histories respectively. α^t denotes a set of agents active at time t and is a subset of all agents, *Agent*. To simplify the discussion, it is assumed that α^t does not change over time.

The system dynamics of π is represented by a predicate $Different(\pi^{t_1}, \pi^{t_2})$ which is defined as follows:

$$Different(\pi^{t_1}, \pi^{t_2}) \leftrightarrow ((s^{t_1} \neq s^{t_2}) \vee (h^{t_1} \neq h^{t_2}) \vee (\alpha^{t_1} \neq \alpha^{t_2}))$$

When the system π changes its state, history, or the set of active agents, $Different(\pi^{t_1}, \pi^{t_2})$ becomes true. In fact, as long as an event occurs between t_1 and t_2, a system π at time t_1 is by definition different from the system at time t_2, that is,

$$Different(\pi^{t_1}, \pi^{t_2}) \leftrightarrow \exists(e, t) \left\{(e, t) \in h^{t_2}, t_1 \leq t \leq t_2\right\}$$

Assume that agents who fire events will observe the corresponding system changes. The system state information of agent a at time t is denoted by $(I_s)_a^t$. This information is *consistent* with the system state at time t_1 (s^{t_1}), denoted by a predicate $T\left((I_s)_a^t, t_1\right)$, iff $(I_s)_a^t$ uniquely identifies s^{t_1}, that is, $(I_s)_a^t \cong s^{t_1}$.

$$T\left((I_s)_a^t, t_1\right) \leftrightarrow (I_s)_a^t \cong s^{t_1}$$

Since an agent is unable to know *a priori* the future state of the system,

$$T\left((I_s)_a^t, t_1\right) \rightarrow t_1 \leq t$$

Notice that there may be several different time points (t_1) at which the predicate $T\left((I_s)_a^t, t_1\right)$ becomes *true*, and they form a set:

$$\left\{t_1 \left| T\left((I_s)_a^t, t_1\right) = true\right.\right\}$$

The function $MaxT\left((I_s)_a^t\right)$ returns the latest t_1 from the set. Thus, an agent a is said to have the *latest system state information* at time t if

$$\forall b \in Agent, MaxT\left((I_s)_b^t\right) \leq MaxT\left((I_s)_a^t\right)$$

Notice that the *latest* state information of the system that an agent has may not be the *most updated* state information of the system, and $(I_s)_a^t$ might not be consistent with the system state at time t. That is, $T\left((I_s)_a^t, t\right)$ can be *false*, although agent a has the *latest system state information* at time t.

5.3 Desired Properties of the Protocol

This section analyzes the desired properties of a distributed protocol that agents execute in order to satisfy requirement **R1**. While an agent executes the protocol, it requires the cooperation of a number of other agents for its successful completion, and thus the protocol is based on the *quorun consensus*. A *quorum* is any such set of agents. The basic idea of *quorun consensus* is that the agent applies for and acquires the permission of multiple agents in a quorum before it can fire an event [171]. In terms of **R1**, it requires that *the quorum only grants the permission to the agent which holds the most updated system state information*. As shown in Figure 5.1, a quorum consists of three agents: Agent2, Agent3 and Agent4. Agent1 must make application for approval of Agent2, Agent3, and Agent4 in order to fire an event.

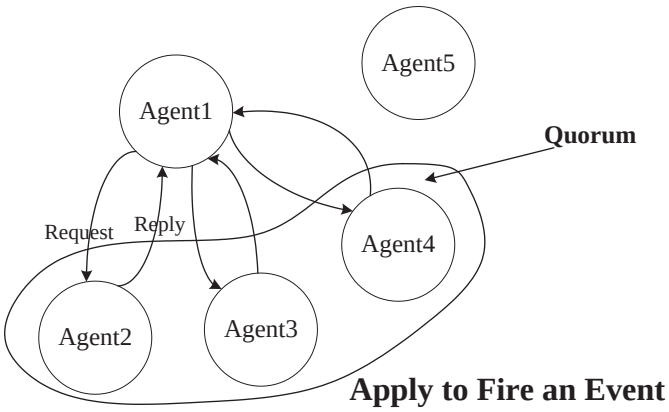

Figure 5.1: Distributed approach based on a quorum consensus.

Let $Agent(A)$ denote the agent who makes the application A, and App is a set of all possible applications. Let $Qrm(A), A \in App$, denote the *quorum* for approving the application A. Assume the time when the quorum $Qrm(A)$ approves A is $Time(A)$, and the set of agents that have approved the application A is $Approve(A), A \in App$. **R1** can be formally stated as follows:

$$R1: \qquad (Qrm(A) \subseteq Approve(A)) \rightarrow T\left((I_s)_a^t, t)\right)$$

As will be proved shortly, the satisfaction of **R1** is amount to satisfying the following three properties:

1. The quorum approves the application of an agent only if that agent has the latest system state information.

2. When the system starts up, all agents know the initial system state which is believed by each agent as the *latest system state information* it has at that time.

3. An agent's system state information evolves over time. In other words, $MaxT\left((I_s)_a^t\right)$ is nondecreasing in time t.

Let t_0 denote the time when system π starts up. Let $Bel_a^t, a \in Agent$, denote the set of agents that are believed by agent a at time t to have the latest system state information. The above three properties can be formalized as

1. $A \in App, a = Agent(A), t = Time(A),\ (Qrm(A) \subseteq Approve(A)) \rightarrow \forall b \in Agent,$
   $MaxT\left((I_s)_b^t\right) \leq MaxT\left((I_s)_a^t\right)$

2. $\forall a \in Agent\left[T\left((I_s)_a^{t_0}, t_0\right) = true, Bel_a^{t_0} = \alpha^{t_0}\right]$

3. $\forall a \in Agent, \forall t_1, t_2, t_1 \leq t_2,\ MaxT\left((I_s)_a^{t_1}\right) \leq MaxT\left((I_s)_a^{t_2}\right)$

These three properties induce the satisfaction of requirement **R1**. It is proved as follows.

Proof: The satisfaction of requirement **R1** can be proved by mathematical induction.

a. When $t = t_0$, and an application $A \in App$ has been approved by the quorum $Qrm(A)$ at time t. That is, $Time(A) = t = t_0$, $Qrm(A) \subseteq Approve(A)$, $a = Agent(A)$. By property 2, $T\left((I_s)_a^{t_0}, t_0\right) = true$. This implies,

$(Qrm(A) \subseteq Approve(A)) \rightarrow T\left((I_s)_a^t, t\right)$.

Meanwhile, if agent a fires an event at time t, at the next time clock, a will still maintain the latest information of the system state.

b. When $t > t_0$, and there is no $(e, t_1) \in h^t$, $t_0 \leq t_1 \leq t$. An application $A \in App$ has been approved by the quorum $Qrm(A)$ at time t. $a = Agent(A)$.

Since there is no $(e,t) \in h^t$, $t_0 \leq t_1 \leq t$, $Different(\pi^{t_0}, \pi^{t_1}) = false$, $s^{t_0} = s^t, h^{t_0} = h^t$.

Because $Time(A) = t$, as $Qrm(A) \subseteq Apprvove(A)$, by property 3,

$MaxT\left((I_s)_a^t\right) \geq MaxT\left((I_s)_a^{t_0}\right)$

As $s^{t_0} = s^t$, $h^{t_0} = h^t$, then $MaxT\left((I_s)_a^t\right) = t$. This implies,

$(Qrm(A) \subseteq Approve(A)) \rightarrow T\left((I_s)_a^t, t\right)$

c. Assume that at the time t, $t > t_0$, there is an application A_1, $A_1 \in App$, that has been approved by the quorum $Qrm(A_1)$, and satisfies the condition

$(Qrm(A_1) \subseteq Approve(A_1)) \rightarrow T\left((I_s)_{a_1}^t, t\right)$,
where $a_1 = Agent(A_1)$, $t = Time(A_1)$.

d. For another application A_2, $A_2 \in App$, if $Qrm(A_2) \subseteq Approve(A_2)$, $Time(A_2) = t_2 > t$, $a_2 = Agent(A_2)$, and there is no $(e,t_1) \in h^{t_2}$, $t \leq t_1 \leq t_2$. By properties 1 and 3,

$MaxT\left((I_s)_{a_2}^{t_2}\right) \geq MaxT\left((I_s)_{a_1}^t\right)$

As $s^t = s^{t_2}$, $h^t = h^{t_2}$, it is easy to check that

$MaxT\left((I_s)_{a_2}^{t_2}\right) = t_2$. This implies,

$(Qrm(A_2) \subseteq Approve(A_2)) \rightarrow T\left((I_s)_{a_2}^{t_2}, t_2\right)$

By mathematical induction, it is therefore proved that the above three properties ensure the satisfaction of requirement **R1**. **Q.E.D**

Properties 2 and 3 can be satisfied trivially. The difficulty in implementing such a protocol is to satisfy property 1. As verified later, property 1 is fulfilled if the following three conditions are ensured.

1.1 The quorum of any two applications must have at least one agent in common.

1.2 An agent a belongs to the quorum approves the application of agent b only when agent b's system state information is consistent with the system at a time, which is later than that of the agent which a believes to have the *latest* state information.

1.3 When an application is approved by the quorum, all agents in the quorum should believe that the agent, who makes the application, has the *latest system state information.*

Let $Time_a(A)$, where $A \in App$, $a \in Qrm(A)$, denote the time when agent a approves application A. Let $Finish(A)$ denote the time when the processing

of application A has been finished (no matter whether it is approved or not). The processing of an application is said finished if all agents involved (the agents belong to the quorum and the agent who makes the application) recognize the result of the application. Finally, let $Start(A)$ denote the time when application A is made. The three conditions 1.1 – 1.3 are re-written as

1.1 $\forall A, B \in App, Qrm(A) \cap Qrm(B) \neq \emptyset$

1.2 If $A \in App, b = Agent(A)$, then

$(a \in Approve(A)) \rightarrow \forall t', Time_a(A) \leq t' \leq Finish(A)$,

$MaxT\left((I_s)^{t'}_{Bel^{t'}_a}\right) \leq MaxT\left((I_s)^{t'}_b\right)$

In fact, we can further regulate that,

$\forall t'', t' < t'' \leq Finish(A), b \in Bel^{t''}_a$

1.3 If $A \in App, b = Agent(A)$, then

$(Qrm(A) \subseteq Approve(A)) \rightarrow \forall a \in Qrm(A)\,(t = Time(A), b \in Bel^t_a)$

The following proposition shows that the obedience of conditions 1.1 – 1.3 ensures the satisfaction of property 1.

Proposition 5 *Property 1 is ensured through satisfying conditions 1.1-1.3, assuming that condition 2' and property 3' are also satisfied.*

In this proposition, property 3', called the *updating property*, is an extension of property 3:

a. $\forall a \in Agent, \forall t_1, t_2, t_1 \leq t_2,\ MaxT\left((I_s)^{t_1}_a\right) \leq MaxT\left((I_s)^{t_2}_a\right)$.

b. $MaxT\left((I_s)^t_a\right) \leq MaxT\left((I_s)^t_{Bel^t_a}\right)$

c. $\forall a \in Agent, \forall t_1, t_2, t_1 \leq t_2,\ MaxT\left((I_s)^{t_1}_{Bel^{t_1}_a}\right) \leq MaxT\left((I_s)^{t_2}_{Bel^{t_2}_a}\right)$

d. When an agent makes no applications, it will not change its system state information. That is,

$$\forall a \in Agent, \forall t_1, t_2, t_1 \leq t_2,$$
$$\neg\exists A \left[\begin{array}{c}(A \in App) \wedge (a = Agent(A), t_s = Start(A),\\ t_f = Finish(A)) \wedge ((t_1 \leq t_s \leq t_2) \vee\\ (t_1 \leq t_f \leq t_2) \vee (t_s \leq t_1 \leq t_2 \leq t_f)\end{array}\right] \longrightarrow \left((I_s)^{t_1}_a = (I_s)^{t_2}_a\right)$$

The updating property is very intuitive and can be easily implemented by the protocol. For example, 3'b basically requires that each agent, who belongs to Bel_a^t, should have later information compared with that of agent a.

Condition 2' in Proposition 5, however, is more technical. its discussion will be postponed until it is exploited in the proof of Proposition refcha-prot-prop1.

Proof of Proposition 5:

Proposition 5 will be proved in two steps. In the first step, the case when agent a belongs to the quorum of an application A ($a \in Qrm(A), b = Agent(A)$) is considered. The second step handles the case when agent a is outside the quorum.

Step 1:

For any agent a, $A \in App$, $a \in Qrm(A)$, if $Qrm(A) \subseteq Approve(A)$, by condition 1.3,

$$MaxT\left((I_s)_b^t\right) \geq MaxT\left((I_s)_{Bel_a^t}^t\right)$$

where $b = Agent(A)$, $t = Time(A)$. By property 3'b,

$$MaxT\left((I_s)_b^t\right) \geq MaxT\left((I_s)_a^t\right)$$

This implies,

$$\forall a, a \in Qrm(A), (Qrm(A) \subseteq Approve(A)) \rightarrow MaxT\left((I_s)_b^t\right) \geq MaxT\left((I_s)_a^t\right)$$

Step2:

For any agent a, $a \in Agent, A \in App, a \notin Qrm(A)$, assume

$$Qrm(A) \subseteq Approve(A), b = Agent(A), t = Time(A).$$

The same result as in *step 1* will be reached by mathematical induction.

a. When $t = t_0$, by property 2, $T\left((I_s)_a^{t_0}, t_0\right) = true$. According to the definition of the function $MaxT$, and the predicate T,

$MaxT\left((I_s)_a^{t_0}\right) = t_0 \leq MaxT\left((I_s)_b^{t_0}\right) = t_0$

Because $t = t_0$, we obtain

$\forall a, a \in Qrm(A), (Qrm(A) \subseteq Approve(A)) \rightarrow MaxT\left((I_s)_b^t\right) \geq MaxT\left((I_s)_a^t\right)$

b. When $t > t_0$, and there is no $(e, t_1) \in h^t$, $t_0 \leq t_1 \leq t$, similar with the previous discussions,

$Different(\pi^{t_0}, \pi^t) = false, s^{t_0} = s^t, h^{t_0} = h^t$.

It is easy to verify that $MaxT\left((I_s)_b^t\right) = t$. As for any agent a, $a \in Agent$, $a \notin Qrm(A)$, $MaxT\left((I_s)_a^t\right) \leq t$. This implies,

$(Qrm(A) \subseteq Approve(A)) \rightarrow MaxT\left((I_s)_b^t\right) \geq MaxT\left((I_s)_a^t\right)$

c. Assume that every application B, which has been approved before time t, satisfies,

$(Qrm(B) \subseteq Approve(B)) \rightarrow$
$\left(MaxT\left((I_s)_c^{t'}\right) \geq MaxT\left((I_s)_{a'}^{t'}\right)\right)$
where $c = Agent(B)$,$t' = Time(B)$,$a' \in Agent$, $t' \leq t$.

d. Assume that there is an agent d, $d \in Agent$, $d \notin Qrm(A)$, such that, $MaxT\left((I_s)_d^t\right) > MaxT\left((I_s)_b^t\right)$. Let $t_1 = MaxT\left((I_s)_d^t\right)$, $t_2 = MaxT\left((I_s)_b^t\right)$, $t_1 > t_2$. By the definition of the function $MaxT$, and the general assumptions about our system, there exists $(e, t_3) \in h^{t_1}$, $t_2 \leq t_3 \leq t_1$. This implies that there is an application B_1, which has been approved by the quorum $Qrm(B_1)$ at time t_3. That is,

$Qrm(B_1) \subseteq Approve(B_1), t_3 = Time(B_1)$.
By the assumption made in part c and condition 1.3, for every agent $d_1 \in Qrm(B_1)$,

$$MaxT\left((I_s)_{Bel_{d_1}^{t_3}}^{t_3}\right) \geq MaxT\left((I_s)_b^{t_3}\right)$$
$$\text{and } MaxT\left((I_s)_{Bel_{d_1}^{t_3}}^{t_3}\right) \geq MaxT\left((I_s)_d^{t_3}\right)$$

By condition 1.1, there exists $d_2 \in Qrm(B_1), d_2 \in Qrm(A)$, such that,

$$MaxT\left((I_s)_{Bel_{d_2}^{t_3}}^{t_3}\right) \geq MaxT\left((I_s)_b^{t_3}\right)$$
$$\text{and } MaxT\left((I_s)_{Bel_{d_2}^{t_3}}^{t_3}\right) \geq MaxT\left((I_s)_d^{t_3}\right)$$

If the assumptions made in part d can induce the existence of an agent d', $d' \in Qrm(A)$, $MaxT\left((I_s)_{Bel_{d_2}^{t}}^{t}\right) > MaxT\left((I_s)_b^t\right)$, a contradiction is then produced. The induction part of the proof will therefore be verified.

1. According to property 3', if during the time period (t_3, t), agent d makes no applications,
   $MaxT\left((I_s)_{Bel_{d_2}^{t_3}}^{t}\right) \geq MaxT\left((I_s)_d^t\right) > MaxT\left((I_s)_b^t\right)$
   This implies that $d_2 \notin Approve(A)$, therefore $Qrm(A) \supset Approve(A)$. A contradiction occurred.

2. Now consider the case that, during the time period (t_3, t), agent d has made an application B_2 at time t_4, $t_4 < t$.

If the application B_2 has been approved before t, $Time(B_2) = t_5$, $t_5 < t$, there exists d_4, $d_4 \in Qrm(B_2)$, $d_4 \in Qrm(A)$, such that

$$MaxT\left((I_s)^t_{Bel^t_{d_4}}\right) \geq MaxT\left((I_s)^{t_5}_{Bel^{t_5}_{d_4}}\right)$$
$$= MaxT\left((I_s)^{t_5}_d\right) = MaxT\left((I_s)^t_b\right) > MaxT\left((I_s)^t_b\right)$$

It therefore leads to $Qrm(A) \supset Approve(A)$. A contradiction occurred.

Generally, if $Qrm(B_2) \supset Approve(B_2)$, $Finish(B_2) = t_5 < t$, then $\forall d_4 \in Qrm(B_2)$,

$$MaxT\left((I_s)^t_{Bel^t_{d_4}}\right) \geq MaxT\left((I_s)^{t_5}_{Bel^{t_5}_{d_4}}\right)$$
$$= MaxT\left((I_s)^{t_5}_d\right) = MaxT\left((I_s)^t_b\right) > MaxT\left((I_s)^t_b\right)$$

For similar reason, it also leads to a contradiction.

If the application B_2 has been approved after t, $Time(B_2) = t_5, t_5 > t$, in this case, in order to ensure the satisfaction of property 1, condition 2' mentioned previously will be utilized.

Condition 2': for any agent a, $a \in Qrm(A) \wedge a \in Qrm(B)$, $A, B \in App$, let $t = Finish(A)$, if $Start(B) \leq t \leq Finish(B)$, then $Time_a(B) \leq t$.

By condition 2', there exists an agent d_5, $d_5 \in Qrm(B_2) \wedge d_5 \in Qrm(A)$, such that $Time_{d_5}(B_2) < t$. By condition 1.2,

$$MaxT\left((I_s)^t_{d_5}\right) \geq MaxT\left((I_s)^t_d\right) > MaxT\left((I_s)^t_a\right)$$

It implies that $d_5 \notin Approve(A)$, $Qrm(A) \supset Approve(A)$. A contradiction occurs. Similar arguments can also be applied to the general case when application B_2 is finished after t. The corresponding discussion is omitted here. Through the above discussions, it is evident that the assumptions made in part d will always lead to a contradiction. It is therefore concluded that

$$(Qrm(A) \subseteq Approve(A)) \rightarrow MaxT\left((I_s)^t_b\right) \geq MaxT\left((I_s)^t_a\right)$$

By mathematical induction (from part a to d), Proposition 5 is consequently proved. **Q.E.D**

In this Section, a total of four properties (properties 1, 2, 3, and 3') and 4 conditions (conditions 1.1, 1.2, 1.3, and 2') have been identified. A protocol that satisfies these properties is presented in the next Section.

5.4 A Protocol for Maintaining the System State Information

In this section, a token-ring-based protocol that satisfies the properties and conditions discussed in the previous Section will be presented. The protocol assumes the *quorum* of each application to be all the active agents excluding one that makes the application. This assumption can be inefficient as every agent has to contribute efforts to each application. However, the protocol is both simple to describe and easy to implement, and is presented here for illustration purposes.

The protocol is based on the use of a token ring. All active agents are logically organized in a communicating ring. Each agent has a predefined position in the ring so that every agent knows who its successor is. There is a token that circulates the ring. It is defined as a tuple of m entries. m is the number of agents. Specifically, a token To is represented as $((e_1, t_1)^1, (e_2, t_2)^2, \ldots, (e_m, t_m)^m)$. Each entry of the token, $(e_k, t_k)^k$, indicates that an event e_k has been fired by agent k at time t_k.

When the system π starts up, a token is allocated to a randomly chosen agent and takes an initial value with each entry being a blank value (NIL). It begins to circulate across the ring. The initial value of the token implies that no agents have fired any events. When an agent wants to fire an event, it waits for token's arrival. This is the behavior of *making an application* for firing an event. Upon receiving the token, the agent examines the token to identify another agent that has fired the latest event (based on the timestamps in the token), and then updates its own information according to the system state information of that agent. In effect, holding a token is an indication that all agents in the ring grant the permission of firing an event to the agent who holds it. Thereafter, the agent fires an event, stores the event information in the corresponding entry of the token, and passes the token to the next agent (giving its implicit approval to other agents). If the agent does not want to fire an event, it simply passes the token to the next agent in the ring. The token-ring mechanism implements the *quorum consensus*.

The protocol is formalized using the state machine approach [245] as follows:

Protocol of agent *i*.
Var_i:
  $busy \in Boolean$, initially *false*
  $(I_s)_i$, system state information of agent i
  t, current system time

Receive_Token: command {token:
    $\left((e_1, t_1)^1, (e_2, t_2)^2, \ldots, (e_m, t_m)^m\right)$ }
  t:=*currentTime*() //synchronized current time.
  if $busy = true$ then
    Send token to the next agent in the ring.
  else if agent i does not want to fire an event then
    Send token to the next agent in the ring.
  else
    Update $(I_s)_i$ by $(I_s)_x$, where x refers to the agent with the latest t_x in the receiving token.
    Fire an event e_t.
    Update the ith *token* entry with $(e_t, t)^i$
    Send token to the next agent in the ring.

Notice that the token is the key to the global knowledge about all agents. The agent, who holds the token and has updated its system knowledge, is believed by all agents in the quorum to be the agent that has the *latest system state information*.

The satisfaction of properties 2 and 3 can be verified trivially. For property 2, it is a direct consequence of the regulation that each entry of the *token* will be assigned a blank value initially. The satisfaction of property 3 can be shown by the fact that agents will have to update their system state information before firing any event. Consequently, as time passes, they may either obtain a more updated system state information or remain as original. However, neither cases violate property 3.

To satisfy property 1, the token-ring-based protocol is shown instead to satisfy conditions 1.1–1.3, property 3' and condition 2'. Since the quorum of any application has $|Agent| - 1 \geq |Agent|/2$ agents, condition 1.1 is ensured. Moreover, because any agent holding the token and firing an event is believed by all agents to have the latest system state information, conditions 1.2 and 1.3 are satisfied. Property 3' also easily holds since an agent's system knowledge evolves over time (3'a) and will be updated when it fires an event (3'd). In addition, 3'b and 3'c are valid because the agent believed by all agents to have the *latest system state information* always has the system knowledge that is more updated than other agents.

As the circulation of the token has imposed an explicit order of making applications among agents, there is no possibility of existing more than one on-going applications (applications that have been made but not been finished yet) within the multiagent system. The precondition of condition 2' is consequently never

valid. From this perspective, the protocol satisfies condition 2'.

By ensuring conditions 1.1–1.3, property 3' and condition 2', property 1 is automatically guaranteed (Ref. proposition 1). By satisfying properties 1 – 3, the token-ring-based protocol therefore achieves requirement **R1**.

5.4.1 Illustration of the Protocol: A Pursuit Game

In order to illustrate the working of the token-ring-based protocol, a well-know *pursuit game* is considered. This game is also used in an experimental study of protocol's effectiveness in the next section. The *pursuit game* has been extensively used as a test case for multiagent reinforcement learning [7, 266]. The learning environment is assumed to be partially observable and agents' perceptual capabilities are restricted.

In a pursuit game, two types of agents, *predator* and *prey*, move concurrently in a grid environment (Figure 5.2). The objective of predators is to catch a prey. They do so by blocking all the possible moving paths of the prey. In Figure 5.2, the four circles in black represent predators. The circle in white denotes a single prey. The predators and prey are modeled as agents, and they are capable of 5 actions, namely, *moving up, moving down, moving left, moving right*, and *staying idle*. The rule of game stipulates that two or more agents cannot possess the same block at the same time, and they cannot move out of the grid border either. The right part of Figure 5.2 represents a final state, where the prey is captured as all its four moving paths are blocked by predators.

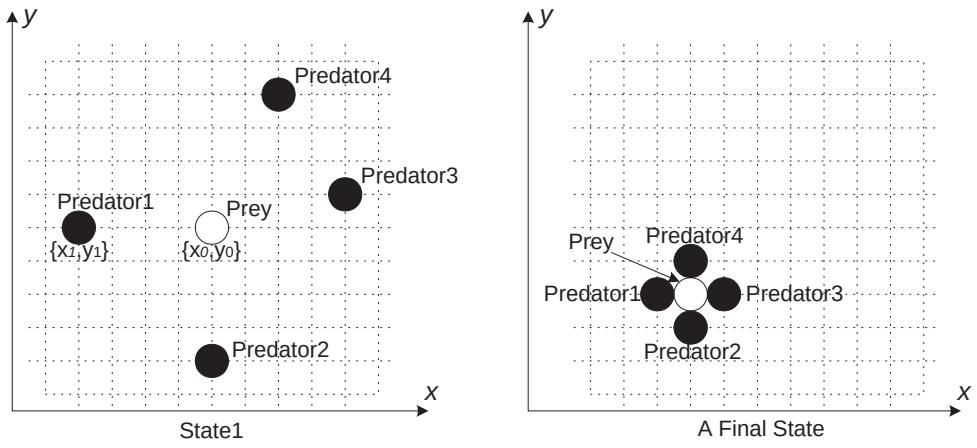

Figure 5.2: The pursuit game with two possible states.

In this game, the prey moves randomly, whereas the predators are assumed learning agents and use reinforcement learning algorithms to decide which action to take. The learning algorithms take two alternative inputs: (1) the current system state as shown in Figure 5.2; or (2) the agents' own observations. With-

out additional help, the predators only have limited perceptual capabilities, that is, they can only see objects (predator or prey) that are at most 2 blocks away (measured in Manhattan distance). Figure 5.3 illustrates the sight of a predator (the shaded area). Using an agent's own observation as inputs, the learning algorithm is said to work in a partially observable environment without the Markov property.

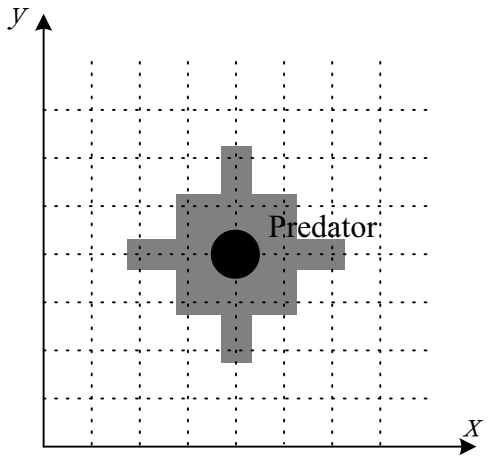

Figure 5.3: The sight of a predator agent.

Now let these agents execute the token-ring-based protocol in order to obtain a global system view that respects the Markov property. Four predator agents and one prey agent form a communication ring as in Figure 5.4, and there is a token circulating the ring. An agent must hold the token before it can take an action. The token records the last actions performed by each agent. Assume that the positions of all agents when the game starts are known *a priori* (Property 2). In this setting, the positions of each predator and the prey constitutes a *global system view*, which is constructed by each agent who intends to fire an event and is used by learning algorithms.

Suppose that a pursuit game starts with a system state shown in the left part of Figure 5.4 and Predator1 is assigned a token. Upon receiving the token, Predator1 knows that the game just starts. It performs an action to move down, records the timestamped action into the token, and passes it to Predator2. If Predator2 intends to fire an event, it recognizes from the token that Predator1 performed the latest action, and thus contacts Predator1 to obtain Predator1's view of the system, i.e., the initial positions of every agent updated with Predator1's own move. At this point, Predator2 has the most updated system knowledge, and is said *approved* by all agents in the ring to take an action. It takes an action, records

that action into the token, and sends the token to Predator3. The same procedure is repeated each time an agent intends to fire an event. As a consequence, the latest system state information can be found via the token and the most updated system state information is always maintained by one agent in the ring.

This illustration shows that the protocol fulfills properties 1, 2, and 3 (Ref. Section 5.3). Particularly, since each predator is able to update its global system view after receiving the token, its application to fire an event is *approvable* by all agents in the ring. Property 1 thus holds.

The use of the token-ring-based protocol is under the assumption that the prey would be willing to give out its position information (to be cooperative). This does suggest that the protocol is best suited to cooperative multi-agent environments [62]. In practice, cooperation may not be realizable in a pursuit game described in this Chapter. Nevertheless, since many reinforcement learning techniques are explored and examined in such games to enable predators to more quickly catch the prey [7], as a comparative study, this game has been chosen as the experimental domain. The real purpose is to use this game to evaluate the effectiveness of the token-ring-based protocol as shown in the next section. Notice that, in domains where agents are inherently cooperative, the protocol can be effective tools that may even guarantee the convergence of the corresponding learning algorithms and therefore effectively improve the learning performance [62].

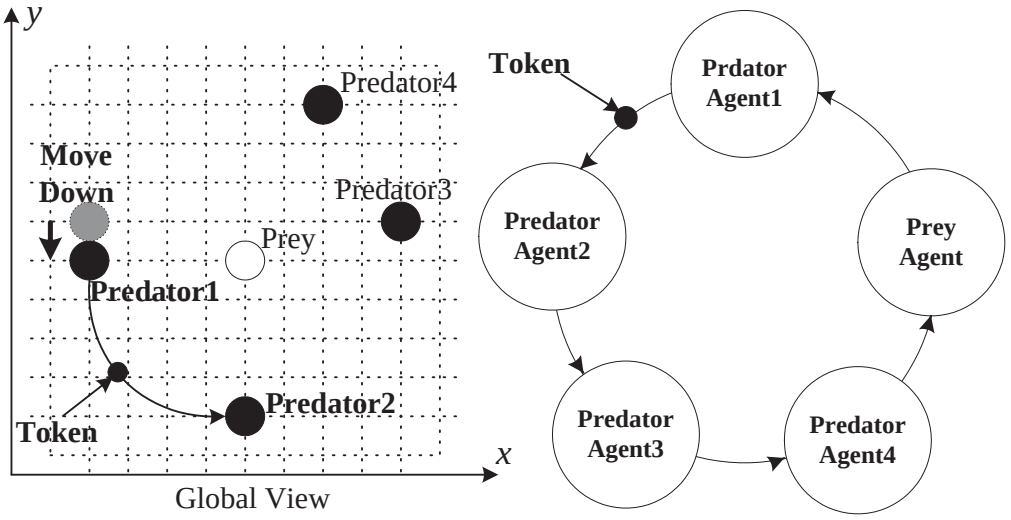

Figure 5.4: A token ring formed by 4 predators and 1 prey.

5.5 Experimental Evaluation

In the experiments described in this Section, agents' perceptual capabilities are restricted as in Figure 5.3, but agents execute the token-ring-based protocol to

obtain a global view. In addition to 1 prey agent and 4 predator agents, the experiment sets up a 7×7 grid environment. The system performance are measured using the number of moves required for predators to catch the prey. The lower the number, the higher the system performance. Each time when the prey is captured, *every* predator receives a reward of value 1. The predator receives no rewards (or a reward of value 0) for other actions.

Two different reinforcement learning algorithms are used to train predators: the *Q-learning approach* [266] and the *profit-sharing approach* [7]. In the Q-learning approach, the quality function is updated each time when a predator performs an action. Each predator maintains its own quality function and learns independently. The decision policy used during the learning process follows the ϵ-greedy method with the probability $\epsilon = 0.3$ [260]. In the implementation of the Q-learning algorithm, the *discounting factor* $\alpha = 0.9$ and the learning rate $\gamma = 0.5$. Figure 5.5 shows two learning curves of the required moves to capture the prey agent, one when the algorithm is supported by the protocol and one is not. Each episode in Figure 5.5 represents an individual pursuit game running from a starting state to an end state where the prey agent is captured.

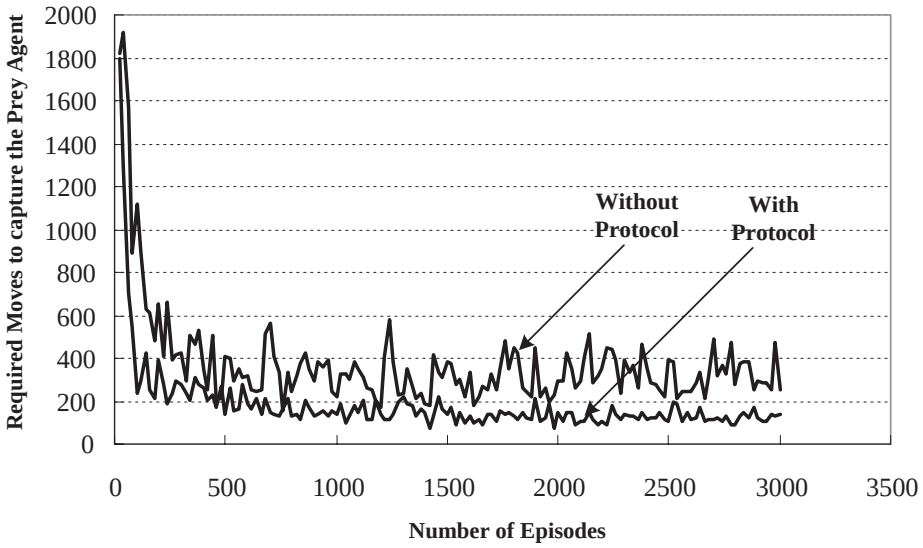

Figure 5.5: The performance of the Q-learning algorithm both with and without our token-ring protocol.

Comparing the two learning curves in Figure 5.5, it is clear that supported by the token-ring-based protocol, the Q-learning algorithm converges more quickly and requires fewer moves to capture the prey. The effectiveness that the protocol brings to the Q-learning algorithm is obvious.

Figure 5.6 compares the system performance after the learning process converges for the cases with and without the protocol. The histograms in Figure 5.6 are obtained by 1000 independent running of the pursuit game. They show that more episodes require less moves to capture the prey agent when the protocol is used. The average number of moves is 67.08. Without the protocol, the number goes up to 234.836. This suggests that predator captures the prey $234.836/67.08 = 3.5$ times faster when executing the protocol.

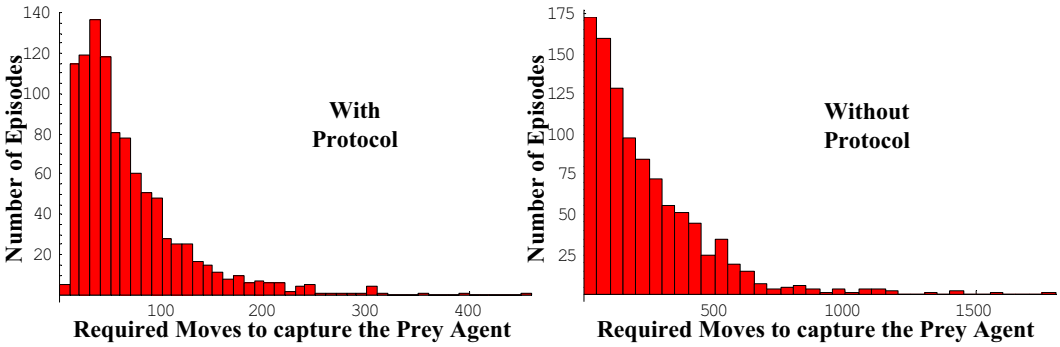

Figure 5.6: The histograms of the Q-learning algorithm both with and without our toke-ring protocol.

Similar experimentation is carried out for the profit-sharing approach [7]. The quality function is updated only when the prey is captured. Each predator agent learns independently and uses the *profit-sharing* learning algorithm to adjust its decision policy. The discounting ratio of the algorithm is set to 0.5 [7]. Figure 5.7 shows the learning curves when the learning algorithm works both with and without the token-ring-based protocol. In addition, the histograms of the performance of learned policies are depicted in Figure 5.8. When the protocol is utilized, the average number of moves to capture the prey agent is 34.288, and the number changes to 115.044 when without it. These numbers suggest that despite of using the profit-sharing learning algorithm, which does not require the environment to be fully observable, the protocol still effectively improves the system performance to $115.044/34.288 = 3.36$ times better. It is also interesting to notice that with the protocol, the Q-learning algorithm outperforms the profit-sharing algorithm when agents learn through their own observations (around $115.044/67.08 = 1.7$ times). These experimental results indeed show that the token-ring-based protocol effectively improves the learning performance.

5.6 Summary

In a multi-agent learning context, it is often desirable that the system maintains the Markov property such that the system behavior only depends on the current system state. In this Chapter, a distributed protocol was presented to ensure that

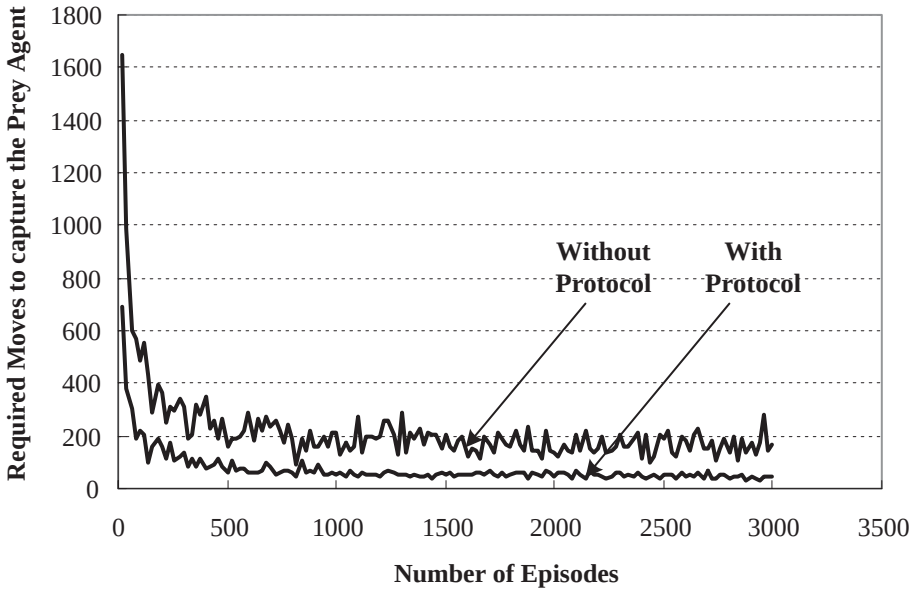

Figure 5.7: The performance of the profit-sharing algorithm both with and without our token-ring protocol.

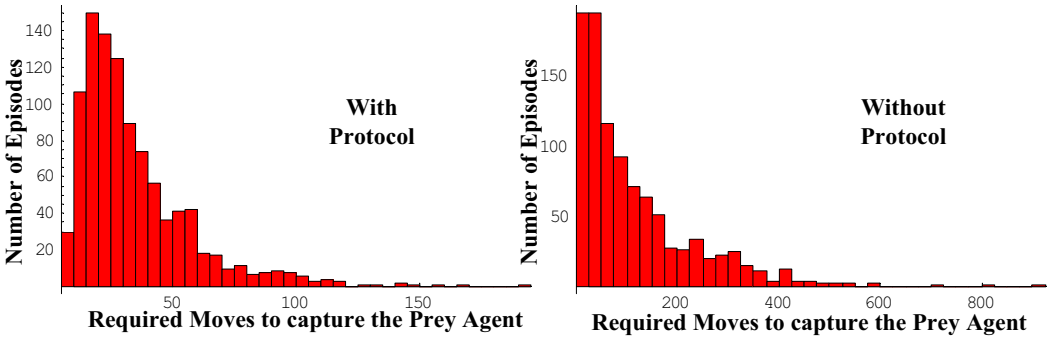

Figure 5.8: The histograms of the profit-sharing algorithm both with and without our toke-ring protocol.

each agent in a system makes their behavioral decisions on the most updated system information. The protocol properties were derived and theoretically analyzed. One token-ring-based protocol was described. The experimental evaluations were conducted for a pursuit game in the context of two learning algorithms. The results show that the protocol is effective and the reinforcement learning algorithms using it perform much better. Besides, the protocol has also been used by the two reinforcement learning algorithms introduced in Chapter 4. The effectiveness of the two algorithms further indicates that the protocol presented in this Chapter can be effective tools that improve the learning and coordination of agents in multi-agent systems.

Chapter 6

Agent Coordination as Dynamic Metabolic Systems: a Biologically-inspired Approach

6.1 Introduction

Coordination in dynamic environments, such as task-oriented domains, is generally approached in either a centralized or a decentralized manner [30]. In the centralized approach, designated coordinator agents are employed to regulate or mediate other agents' activities [68, 162]. Such approach is sometimes inefficient as it can easily become a performance bottleneck to manage centrally the complex interdependencies among all participating agents [203]. In the distributed or decentralized approach, each agent makes its local coordination decisions based upon the current system dynamics [196]. The decentralized approach is quite robust to unexpected events, while at the same time achieving near-optimal global performance [13, 47, 184, 238]. In Chapters 4 and 5, decentralized coordination approaches based on reinforcement learning methods have been extensively investigated. Without focusing only on this single class of techniques, in this Chapter, a *metabolic coordination approach* (in fact, this coordination approach can also be used centrally by designated coordinator agents) inspired by biological metabolic systems will be presented [61]. With this approach, the identification of proper coordination decisions is viewed as an *adaptive* process. A dynamic coordination model is proposed to establish the analogy between task-oriented domain and biological metabolic system, which serves as a good example of effectively coordinated systems.

The basic modeling ideas are derived from the correspondence between task-

oriented domains and metabolic systems. *Tasks* are represented as a *metabolic network* that links multiple biochemical substances under the effect of *enzymes*. Agents' local operations are represented by enzymes. Agents' decisions to perform any operations are made in a *stimulus-response* manner [32]. Agent coordination is viewed as a *metabolic regulation process* through which the system dynamics exhibit certain desirable properties. In this metabolic coordination approach, dynamic optimization algorithms are explored to adjust agents' local decisions (i.e. their response thresholds). The local adjustments are performed to satisfy collectively certain system constraints and obtain expected global performance [28, 42].

In AI, the biology metaphor for solving problems has become a hot topic recently [32, 64, 73, 75, 96]. This approach emphasizes distributedness, allowing agents with limited reasoning capabilities to follow simple decision strategies and to collectively form a flexible, efficient, and robust ensemble. More and more researchers are interested in this new exciting way of solving large-scale real-world problems [51,98,246,276]. Inspired by the many successful research works reported in the literature, Chapter 6 seeks to introduce a new biological analogy, which will be used to tackle a complex shop floor controlling problem. The proposed metabolic system model paves the way for a direct application of a run-time adjustment algorithm. Using this algorithm, agents are able to self-tune their behaviors in the direction of improving the global system performance. Comparing with the Genetic Algorithm, which is another bio-inspired heuristic technique, at least two main advantages of the metabolic system approach in the context of shop floor controlling are discernible.

- The metabolic system approach emphasizes the autonomy concept that underpins the agent technology. With this approach, agents take a full control of themselves. The global system behavior is thought to be emergent from agents' local decisions and interactions. Distributed functioning replaces control, preprogramming, and centralization. On the contrary, genetic algorithm is more ideal for a centralized application where designated coordinator agents are responsible for identifying good decision strategies for other agents.

- In comparison with the genetic algorithm, the metabolic system approach is more computationally efficient from a distributed system point of view. Each agent adjusts its own behaviors, which can be seen equivalent to solving a confined dynamic optimization problem. The computational complexity for each agent to solve the problem is comparatively lower than the process of solving a global optimization problem using the genetic algorithm. In

this sense, the metabolic system approach helps making a more balanced use of the computational resources of all agents. It is worth to note that the benefits brought forward by this distributed approach are provided for free without sacrificing the system performance. As evidenced in the experiment study of this Chapter, the system performance using the two approaches (i.e. the metabolic system approach and the genetic algorithm) is pretty much the same.

In addition to the two advantages listed above, another two potential benefits of using the metabolic system metaphor may also deserve a brief description.

- The metabolic system metaphor presents a simple agent interaction pattern, which is ideal for agents that have only limited reasoning capabilities. Agents in the metabolic system make their local decisions using the so-called response-threshold mechanism, which can be expressed compactly as an easy-to-compute mathematical formula.

- The metabolic system metaphor helps abstracting the global system dynamics in the form of discrete system transition functions. Such functions intend to capture the up-to-date system dynamic information and are utilized explicitly by each agent to adjust its local decisions. This amounts to coordinating agents from a dynamic systems perspective. In the literature, although agent coordination problems are successfully analyzed using the dynamic systems theory [21, 94], few mechanisms have been designed to leverage on this opportunity.

The remainder of this Chapter is organized as follows. Section 6.2 summarizes and compares related research. In Section 6.3, the metabolic system model is introduced, and the agent coordination is presented as a metabolic regulation process. Section 6.4 describes our coordination approach in detail followed by experimental evaluations of its effectiveness. The practicality of the proposed coordination approach is assessed in Section 6.5 for a shop floor application. Finally, Section 6.6 concludes this Chapter.

6.2 Related Work

As described in Chapter 2, a lot of work has been done in the context of multi-agent coordination. Experience tells us that there is no solution that suits all situations [104, 106]. In order to coordinate agents well, specific domain knowledge should be explored. Even in a well-specified environment, such as a task-oriented domain, different requirements may lead to distinct solutions. For example, in

Internet information gathering applications [204], tasks have complex interdependencies. To facilitate cooperation among different tasks, *top-down* coordination algorithms such as GPGP seem suitable [91]. These techniques assume that agents have strong reasoning capabilities and sophisticated mental states, a practical assumption as information retrieval agents often operate in powerful servers and are supported by scheduling or planning sub-systems [262]. In this chapter, a different situation in task-oriented domains will be considered. Specifically, we seek to coordinate a group of agents serving as controllers in shop floor applications. Agents have limited computation capabilities, which render complex planning or scheduling techniques useless. They must be simple with restricted mental states, normally responding system changes *reactively* [41].

There has been much research into coordinating "*shop floor*" agents recently. One popular coordination paradigm is that of market systems. In the various market-based approaches, agents represent resources. The assignment of resources to tasks are mediated by bidding procedures. Evidence abounds, indicating that market-based approaches may achieve near-optimal performance [49, 79, 129, 183, 195, 223]. As Walsh *et. al.* shows, in many of these systems, an equilibrium solution is an optimal solution [282]. However, equilibrium solutions may not always exist and finding them could be NP-hard. For many bidding strategies designed out of human intuition, there lacks useful insight (especially theoretical insight) of their success in most applications. The situation becomes even tougher when system behaviors have to respect other manufacturing constraints. In worst cases, as Kempf and Beaumariage show, a distributed shop floor system can exhibit chaotic behavior in the presence of even simple decision policies [158].

Another coordination approach that has been gaining popularity draws largely from biological analogy. This approach emphasizes certain characteristics such as distributedness, direct or indirect interactions, flexibility, and robustness [32]. The number of its successful applications is exponentially growing in last decades [32]. For example, many shop floor scheduling systems are being built from the Ant Colony Optimization (ACO) method of Dorigo *et. al.* [96, 97]. ACO utilizes a population of simple, ant-like agents, interacting indirectly through laying and following pheromone in the solution space, to identify proper schedules [276]. Although a population of agents is involved in solving coordination problems, all solutions are normally built in advance. In essence, only *static* problems rather than *dynamic* problems are considered by this type of approaches. Several attempts of using ACO to solve dynamic coordination problems have been reported [73, 246]. However, they suffer from frequent convergence to undesirable solutions. Compared with these approaches, our coordination approach constantly adapts agents'

local decision policies to the changing environment. The system performance is gradually improved and other manufacturing constraints are also satisfied.

In [270], Theraulaz *et. al.* present a model for the self-organization and coordination that take place within a colony of wasps. They model the colony's coordinated allocation of tasks using what they refer to as response thresholds. A wasp agent has a response threshold for each resource it maintains. Based on a wasp's threshold and the environment stimulus, which is in the form of coming tasks, it may or may not allocate its resources to these tasks. Such interactions between members of the colony and the their environment result in dynamic distribution of tasks [72]. In order to achieve good system performance, various strategies have been proposed to adjust wasps' thresholds. In [33], Bonabeau *et. al.* adopt a model where thresholds remain fixed over time (thresholds are optimized in advance). But in [75, 269], a threshold for a given task decreases during time periods when that task is performed and increases otherwise. Most of the threshold updating strategies are built from heuristics. There is no clear connection between their function and the system performance. Recently, wasp-like agents have been shown to be of great use in the self-coordination of task assignment in shop floor applications [51,74,75]. For example, Cicirello and Smith designed a wasp agent system for vehicle paintshop booth assignment. Their system is shown to be competitive to the "real-world proven" marked-based truck painting system [194].

Researchers may have good reasons to find biology-inspired coordination approaches appealing [32]: at a time when a system is populated with agents that have limited computation capabilities and mental states, and when the application desires a distributed implementation where agents make coordination decisions by themselves (i.e. not always through a centralized controller or coordinator). From an application perspective, biology serves as a new kind of analogy to comprehend the complexity of multi-agent systems. It provides a competitive approach to the market-based coordination paradigm. In particular, these two types of approaches are often used together in shop floor environments. For example, in [75], wasp agents use response thresholds to decide whether to bid or not to bid based on task type. Campos *et. al.* use the same mechanism to adjust bid determination rules [51].

Despite of promising results, in many biology-inspired works, such as Cicirello and Smith's wasp agent system, each task requires an agent to perform one domain operation and all agents are of equal capabilities [75]. This chapter seeks to tackle this limitation and provides a new biological analogy of the shop floor environment, which is based on metabolic systems. The metabolic system model in this Chapter allows several domain operations to be performed by multiple agents

with different capabilities in order to complete a single task (i.e. tasks have more complicated structures). Our metabolic approach goes beyond the simple analogy by systematically adjusting agents' thresholds through dynamic optimization algorithms, while only heuristic updating strategies have been adopted by Cicirello and Smith. Although our system implementation provides a distributed adjustment scheme, agents' thresholds can also be updated centrally by designated coordinator agents. In essence, agents may be organized into several groups and each group is managed by a separate coordinator. In shop floors containing a large amount of agents, this organization may reduce the coordination burden of those agents that work as controllers.

Dealing with complex task structures is not unique to our approach. For example, Walsh and Wellman proposed a *task dependency network* to model hierarchical tasks in a supply chain [279, 280]. The nodes of the network represent agents and goods. The edges in the network represent input/output relationships between agents and goods. Each agent produces exactly one type of goods with pre-determined manufacturing costs. Agent coordination aims at optimizing the allocation of resources in order to maximize the difference between the sum of consumer value and sum of supplier cost [280]. The problem of finding optimal resource allocations is NP-complete. To avoid this complexity, Walsh and Wellman utilize the market coordination paradigm and a *price system* to find near optimal solutions [279]. Later development analyzes this market protocol in the context of competitive equilibria and demonstrates its effectiveness [281]. Different from their research, our metabolic system model considers agent coordination within a shop floor, which is treated as a single agent in the task dependency network. Each agent in our model produces multiple types of goods. In addition to improving the system performance, other manufacturing constraints are also respected. Without relying on the existence of competitive equilibria, the metabolic system model is used to provide a high-level description of system dynamics, which is used by agents to adjust their local decisions. It is to be noted that no intention is taken in this Chapter to replace the market coordination paradigm, which has been extensively evaluated in fields like supply chain management. In fact, as a promising direction of future research, our metabolic approach may be used by supply chain agents to improve their decisions in a market environment.

Alternative methods, such as the Markov Decision Process (MDP) [288], have also been used to model shop floor environments. In Chapter 4, we show that a coordination problem modeled with MDP can be solved by reinforcement learning methods. The system performance is evaluated as a weighted summation of the values of a reward function at each system state along the system trajectory [37]. This could become a restrictive requirement when the system behavior is better

evaluated by a non-linear function and when there are other manufacturing constraints to be satisfied. From a theoretical perspective, the two types of models (i.e. dynamic model as our metabolic system and the MDP model) correspond to the two main tools of dynamic optimization: *calculus of variations* and *dynamic programming* [42]. Industrial applications, such as the field of optimal system control, favor the former approach as calculus of variations has been more extensively explored with numerous successful applications [42,154,157]. Part of the initiative of this Chapter is to introduce this class of techniques to the field of multi-agent coordination. It is expected that experiences gained from the industrial systems could lead to much more effective and practical coordination solutions. Table 6.1 details further the relationships between the learning approach introduced in Chapter 4 and the dynamic coordination approach introduced in this Chapter.

6.3 Metabolic System Model for Dynamic Agent Coordination

In this section, the biological metabolic system model is introduced to the problem of dynamic agent coordination in shop floor applications. The modeling concepts are presented first, followed by analysis and illustration. Instead of strictly obeying real-life biological facts, the modeling treatment gives enough concepts through which the global system dynamics can be formalized.

6.3.1 Metabolic System Model

In a task-oriented domain, an agent's activity is to achieve a set of *tasks*. In shop floor applications, a task is to produce a specific product. It requires one or more *cooperative* agents to perform multiple manufacturing operations. In order to describe the high-level system dynamics, an analogy between shop floor system and the metabolic system is established through the metabolic system model.

In a nutshell, a metabolic system is a dynamic biochemical reaction and exchange system, and is comprised of numerous biochemical substances spread over all metabolic organs (for convenience, these organs are called agents). Every agent has its own interior repository. It performs continuously three processes in sequence, the *absorption*, *release*, and *transformation* processes. Through the absorption process, exterior biochemical substances are moved into an agent's interior repository. Conversely, by the release process, those substances contained in the agent are transferred to other agents or to the environment. The transformation procedure takes place within the interior repository and transforms one or several biochemical substances into other substances under the mediation of proper enzymes. An agent's capability is determined by the types of enzyme it

		Learning Approach	Dynamic Approach
General Application Domain		Task-oriented Domain	Task-oriented Domain
Domain Modeling	Task	Supported	Supported
	Hard Relation	Supported	Supported
	Soft Relation	Supported	Not Supported
	Meta Method	Supported	Not Supported
	Relation Representation	Hard relation graph and soft relation graph	Metabolic Network
	System History	Episodic, each episode contains a limited set of tasks	Constantly involving, the number of tasks is unlimited.
	Reward Function	A weighted summation of the values of a reward function at each system state	Any function (e.g. nonlinear) defined over the system states along the system trajectory.
	Global System Constraints (defined over system states)	Not Supported	Supported
	System Transition	Probability approach, the system transition satisfies the Markov property	Dynamic function that models the density changes of biochemical substances for each agent
	System Dynamics	Discrete	Discrete
Decision Making Mechanism		Fuzzy rules based representation of the quality function	Response threshold mechanism
Decision Adjustment Mechanism		Reinforcement Learning Algorithm	Dynamic Optimization Algorithm
Application Summary		Task-oriented domains that are of an episodic nature. The domain methods have intricate inter-relations and the system performance is measured as a linear summation of the performance value at each system state.	Task-oriented domains that are constantly evolving. The domain methods are linked by only hard relations. The global constraints defined over the system states are considered. The system performance is evaluated as a nonlinear function of the system states.

Table 6.1: The Relationship between the Learning Approach and the Dynamic Coordination Approach.

can offer. It makes use of these enzymes to affect and even control the metabolic system.

An enzyme can mediate only one biochemical transformation. To simplify the interactions between agents and their environments, the following assumptions will be taken: (1) only biochemical substances that cannot be produced by any metabolic agents can be absorbed from the environment; (2) only biochemical substances that cannot be further transformed by any metabolic agents can be released to the environment; and (3) biochemical substances that cannot be released to the environment must be absorbed by other metabolic agents upon their release.

Biochemical substances are linked together via various enzymes and constitute a *metabolic network* [131]. An example metabolic network is depicted in Figure 6.1 in which nodes denote the type of biochemical substances. For instance, CSI_1, CSM_1, and CSO_1 represent three types of biochemical substances. Enzymes E_1 and E_3 link these substances and appear as labels of the respective arrows. The numbers at both edges of an arrow give the relative ratio of biochemical substances in a transformation. For example, one CSM_1 and one CSM_2 will be transformed into one CSO_1 under the effect of an enzyme E_3.

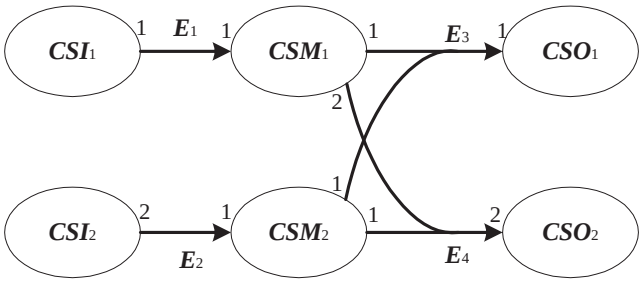

Figure 6.1: A simple metabolic network.

A metabolic network is used to represent tasks in a shop floor system. Producing a *product* denoted by either CSO_1 or CSO_2 is considered as a task. These products are to be manufactured from *parts* CSI_1 and CSI_2, which will be intermediately transformed into *semi-manufactured* products CSM_1 and CSM_2. The manufacturing operations to be performed during this procedure are represented by enzymes that establish the corresponding transformation links and are offered by shop floor agents.

The dynamics of a metabolic system is described through the change of biochemical substances. Suppose that any agent's interior repository has fixed size. Let $\vec{x}_i$ be a *density vector* representing the densities of biochemical substances contained in $Agent_i$. The *combined density vector* $\vec{X}$ thus characterizes the current *state* of a metabolic system.

$$\vec{X} = (\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_n)^T$$

where n is the number of agents in the system. $\vec{x}_i$ evolve at discrete time steps [167]. The change of $\vec{x}_i$ between two consecutive time steps is determined by the combination of three functions: the *absorption* function $Ab^i(\cdot)$, the *transformation* function $Tr^i(\cdot)$, and the *release* function $Re^i(\cdot)$. After incorporating the changes, the density vector $\vec{x}_i$ must remain *valid*. The summation of all entries of $\vec{x}_i$ is no more than 1 and every entry of $\vec{x}_i$ is between 0 and 1. From a system perspective, the three functions $Ab^i(\cdot)$, $Tr^i(\cdot)$, and $Re^i(\cdot)$ form the discrete *dynamic model*

of $Agent_i$. Knowing each agent's dynamic model, the global system dynamics are represented by a *system function* $Sys(\cdot)$, which relates the combined density vector $\vec{X}$ between any two consecutive time steps t and $t+1$ according to

$$\vec{X}(t+1) = Sys\left(\vec{X}(t)\right)$$

For any agent ($Agent_i$), it is convenient to model the release and absorption processes through the *density diffusion* mechanism. This mechanism is represented directly with the following equation

$$\vec{y}_i(t+1) = -1 \cdot M_i \times \vec{x}_i(t) \tag{6.1}$$

where M_i is a diagonal matrix with its j-th diagonal element denoted as $k_j^i(t)$, $0 \leq k_j^i(t) \leq 1$. $k_j^i(t)$ is a *release ratio* at time step t. It is greater than 0 if biochemical substances indexed by j can be released by $Agent_i$. Equation (6.1) establishes a proportional relation between the amount of biochemical substances released and their respective densities. Notice that as $k_j^i(t)$ can take various values, it is possible that higher density substances may have a lower probability for release/absorption at time step t.

In a metabolic system, an agent has a response threshold for each type of enzymes it can offer. Based on these thresholds and its density vector, which serves as the stimulus, the agent offers a certain amount of enzymes of each type. A lower response threshold for a particular type of enzymes contributes to a higher likelihood of offering more such enzymes for same stimulus. Suppose that $Agent_i$ is capable of offering m different types of enzymes, its set of response thresholds is denoted as

$$\Theta_i = \left\{\theta_i^{E_1}, \theta_i^{E_2}, \cdots, \theta_i^{E_m}\right\}$$

The corresponding enzyme set is

$$\mathcal{E} = \{E_1, E_2, \cdots, E_m\}$$

$\theta_i^{E_j}$ is the *response threshold* of $Agent_i$ for offering enzyme E_j. For any enzyme E_j, there exists a set of vectors, V^{E_j}. Each vector $\vec{v} \in V^{E_j}$ is of the same dimension as the density vector $\vec{x}_i$, and all entries of $\vec{v}$ are nonnegative. $\vec{v}$ gives a particular source of stimulus for offering E_j. Use the metabolic network in Figure 6.1 as an example, the densities of both CSM_1 and CSM_2 can stimulate $Agent_i$ to offer enzyme E_3. Therefore, V^{E_3} contains two members $\vec{v}_1$ and $\vec{v}_2$. All entries of $\vec{v}_1$ are 0 except one entry related to the density of CSM_1, which equals to 1. $\vec{v}_2$ takes the same form whereas the entry that takes value 1 is related to the density of CSM_2. Based on V^{E_j}, the total stimulus for $Agent_i$ to offer E_j is defined as

$$S_i^{E_j} = \sqrt[p]{\frac{1}{\sum_{\vec{v} \in V^{E_j}} \left(\frac{1}{\vec{v}^T \times \vec{x}_i} \right)^p}} \tag{6.2}$$

where p is a positive integer. Two important facts apply to Equation (6.2). First, for identical $\vec{x}_i$, Equation (6.2) returns a stimulus that is always smaller than

$$min \left(\left\{ \vec{v}^T \times \vec{x}_i \middle| \vec{v} \in V^{E_j} \right\} \right) \tag{6.3}$$

For significantly large p, the difference between Equations (6.2) and (6.3) can be arbitrarily small. In all of our experiments, p is set to 10. For example, when the densities of CSM_1 and CSM_2 contained in $Agent_i$ are 0.5 and 0.4 respectively, the difference between $S_i^{E_3}$ and 0.4 is only 0.004. In most situations, $S_i^{E_j}$ therefore serves as a good approximation of Equation (6.3). Second, the stimulus function is *continuously differentiable* except when $\vec{v}^T \times \vec{x}_i = 0$. In that case, the stimulus is set to zero. Our definition of $S_i^{E_j}$ makes agents' dynamic models differentiable, which is important in order to use dynamic optimization methods such as *calculus of variations* [42]. For any enzyme $E_j \in \mathcal{E}$, the probability for $Agent_i$ to offer E_j is

$$Pr^i(E_j) = \frac{I_j}{\sum_{I_k \in \mathcal{I}_i} I_k} \tag{6.4}$$

There is a one-to-one correspondence between $I_k \in \mathcal{I}_i$ and $E_k \in \mathcal{E}$. I_k is evaluated according to the *stimulus response mechanism* for adaptive behaviors [269,290] as

$$I_j = \frac{\left(S_i^{E_j}\right)^2}{\left(S_i^{E_j}\right)^2 + \left(\theta_i^{E_j}\right)^2} \tag{6.5}$$

The higher the stimulus $S_i^{E_j}$, the higher the I_k, and therefore the more probable $Agent_i$ will offer enzyme E_j. It is easy to see that

$$\sum_{E_j \in \mathcal{E}} Pr(E_j) = 1$$

Suppose that for each transformation involving one E_j, p biochemical substances CS_p are consumed and transformed to q CS_q, the density change of CS_p is

$$-p \cdot k_i^{E_j}(t) \cdot Pr^i\left(E_j\right)$$

and the density change of CS_q is

$$q \cdot k_i^{E_j}(t) \cdot Pr^i(E_j)$$

where $k_i^{E_j}(t)$ is a *reaction factor* that measures the efficiency of $Agent_i$ in performing transformations mediated by E_j at time step t.

In the metabolic system model, agent coordination is viewed as a *metabolic regulation* process, through which the system exhibits desirable properties. For many shop floor applications, it is appropriate to represent these properties by a constrained dynamic optimization problem, which is stated as

$$min(J) = min\left(\Phi\left(\vec{X}(t_0+N)\right)\right) \tag{6.6}$$

subject to

$$\vec{X}(t+1) = Sys\left(\vec{X}(t)\right), t \in \{t_0, t_0+1, \cdots, t_0+N-1\}$$
$$\text{and } g\left(\vec{X}(t_0+N)\right) \leq 0$$

where J is a real-valued *performance index*. The function $g(\cdot)$ maps any combined density vector $\vec{X}$ to another vector of dimension g, and resembles the g inequality (manufacturing) constraints. The positive integer N is domain dependent and can take any appropriate values. The time step t_0 is not fixed and refers to any time step as long as the metabolic system still evolves. The agent coordination is an on-going process, and at each time step the optimization problem given in Equation (6.6) is expected to be solved satisfactorily.

6.3.2 A Simple Metabolic System

A metabolic system involving an encounter of two agents is presented in this subsection for demonstration purposes. Each agent is able to perform all transformations given in Figure 6.1[1]. Biochemical substances CSI_1 and CSI_2 are provided from the outside, while CSO_1 and CSO_2 produced will be released to the environment. CSM_1 and CSM_2 are also exchanged between the two agents during each time step (Figure 6.2).

Let

$$\Theta_1 = \left\{\theta_1^{E_1}, \theta_1^{E_2}, \theta_1^{E_3}, \theta_1^{E_4}\right\} \text{ and } \Theta_2 = \left\{\theta_2^{E_1}, \theta_2^{E_2}, \theta_2^{E_3}, \theta_2^{E_4}\right\}$$

be the set of response thresholds of $Agent_1$ and $Agent_2$, respectively. An agent's dynamic model, particularly the absorption, release, and transformation processes, follows exactly the discussions given in the previous Subsection. Specifically for our simple metabolic system, the density change of biochemical sub-

[1]Notice that there is no one-to-one relation between nodes of a metabolic network and shop floor agents. An agent may contain any biochemical substances as determined by applications.

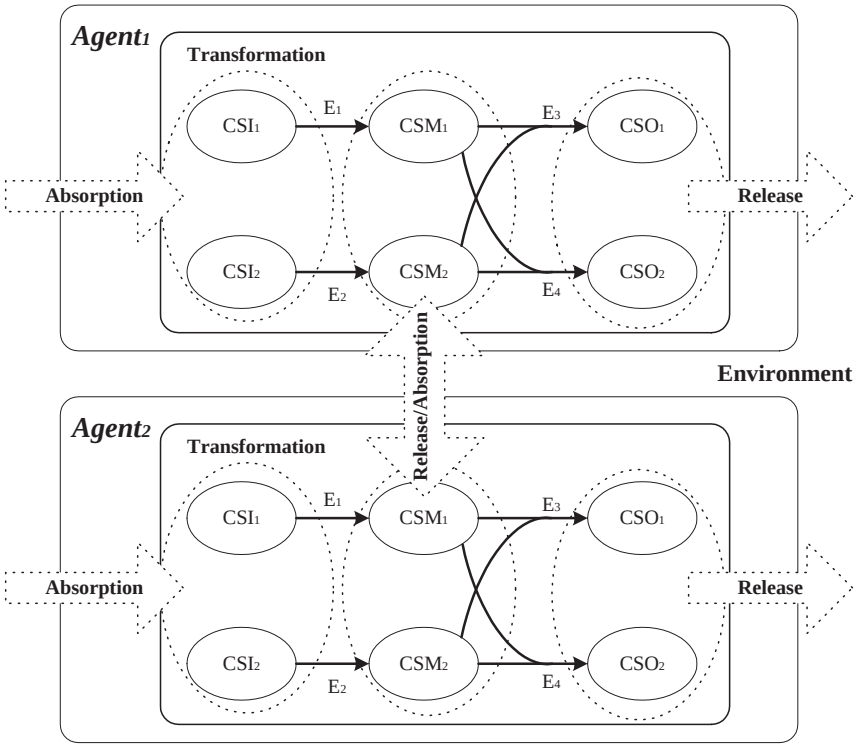

Figure 6.2: An example metabolic system.

stances, such as CSM_2 of $Agent_1$ due to the transformation mediated by E_4, is

$$-k_1^{E_4} \cdot \sqrt[p]{\frac{1}{\left(2/Den^1(CSM_1)\right)^p + \left(1/Den^1(CSM_2)\right)^p}} \cdot Pr^1(E_4)$$

$Pr^1(E_4)$ is obtained by Equations (6.4) and (6.5). $Den^1(\cdot)$ returns the density of any biochemical substances contained in $Agent_1$. $k_1^{E_4}$ is a constant. The first two factors stand for the reaction factor $k_1^{E_4}(t)$ at time step t. Because two CSM_1 will participate in each transformation mediated by one E_4, the number "2" appears in the denominator of the above formula (Ref. Figure 6.1). Thus, the density changes of CSM_2 is upper bounded by the minimum of $Den^1(CSM_1)/2$ and $Den^1(CSM_2)$. It is trivial to show that if $0 \leq k_1^{E_4} < 1$, the resulting density vector $\vec{x}_1$ after applying this transformation process remains *valid*.

For simplicity, we assume that all CSO_1 and CSO_2 produced at each time step are released to the environment. For $Agent_1$ (the same applies to $Agent_2$), the density change of CSI_1 and CSI_2 after the absorption process (the release process is inapplicable) is

$$k_{re}^1 \cdot \frac{Pr^1(E_1)}{Pr^1(E_1) + Pr^1(E_2)} \cdot Fr_1 \text{ and } k_{re}^1 \cdot \frac{Pr^1(E_2)}{Pr^1(E_1) + Pr^1(E_2)} \cdot Fr_1$$

where Fr_1 is the *free space* of $Agent_1$. The first two factors of each formula give the absorption(release) ratio at time step t. Due to the fraction terms, biochemical substances that are more probable to be transformed will have a higher absorption

ratio. The evaluation of the density change of CSM_1 and CSM_2 is almost the same except for the interactions between the release and absorption processes. That is, CSM_1 or CSM_2 released by one agent will be absorbed by the other agent. The details will not be covered further.

Upon determining each agent's dynamic model, the global system dynamics are consequently obtained in terms of the system function $Sys(\cdot)$. In the following, numerical analysis is performed to understand global system behaviors. In all studies,

$$k^{E_1} = k^{E_2} = k^{E_3} = k^{E_4} = 0.2, k_{re} = 0.3, \text{ and } p = 10$$

for both $Agent_1$ and $Agent_2$[2]. Figure 6.3 shows the dynamics of the metabolic system when the system starts evolving from two varied combined density vectors ($\vec{X} = (\vec{x}_1, \vec{x}_2)^T$). The two agents use fixed response thresholds as given in Table 6.2.

$\theta_1^{E_1}$	$\theta_1^{E_2}$	$\theta_1^{E_3}$	$\theta_1^{E_4}$	$\theta_2^{E_1}$	$\theta_2^{E_2}$	$\theta_2^{E_3}$	$\theta_2^{E_4}$
0.2	0.001	1.0	1.0	0.2	0.01	1.0	1.0

Table 6.2: Response thresholds taken by $Agent_1$ and $Agent_2$.

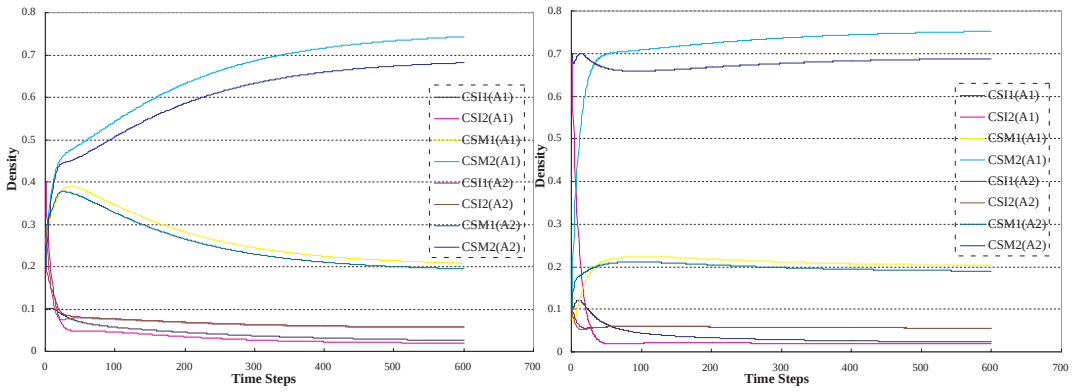

Figure 6.3: System dynamics with varied initial $\vec{X}$.

Figure 6.3 gives the density change of biochemical substances contained both in $Agent_1$ and $Agent_2$. It indicates that although starting with varied combined density vectors $\vec{X}$, the metabolic system converges to the same *fixed point*, which is a solution of the equation

$$\vec{X} = Sys\left(\vec{X}\right) \tag{6.7}$$

Actually, the modulus of all eigenvalues of the corresponding *Jacobian matrix* at this fixed point is less than 1 (the largest modulus is around 0.642). By

[2]For other modeling parameters, similar system dynamics are observed, as evidenced by numerical analysis

Lyapunov theorem, this fixed point is *stable* [95]. For any possible modeling parameters of the metabolic system, there exists at least one *internal* fixed point. *Internal* means that all entries of $\vec{X}$ at the fixed point are less than 1. This fact is a direct application of *Brouwer's Fixed Point theorem* [198]. It is also straightforward to identify another two fixed points. At one point ($\vec{X}_a$), the densities of CSM_1 of both $Agent_1$ and $Agent_2$ are 1, and at the other point ($\vec{X}_b$), the densities of CSM_2 of the two agents are 1. In case when the response thresholds are fixed as in Table 6.2, the two fixed points are *unstable*. Nevertheless, when the responses thresholds take different values, they are capable of becoming stable. Figure 6.4 illustrates such two cases when the system dynamics converge to these two fixed points respectively.

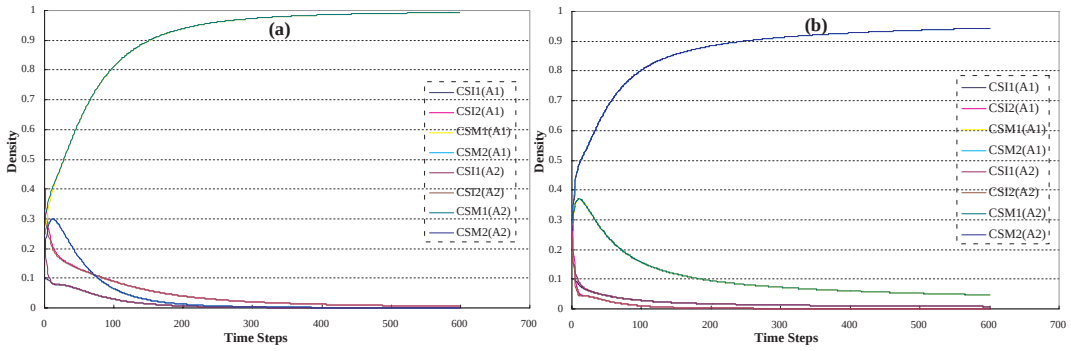

Figure 6.4: System dynamics that converge to different fixed points.

Figure 6.4(a) shows the situation when the system dynamics converge to $\vec{X}_a$. This is achieved by letting $\theta_1^{E_2} = \theta_2^{E_2} = 0.2$ while keeping other response thresholds as in Table 6.2. Figure 6.4(b) gives another situation where the fixed point $\vec{X}_b$ becomes *attractive*, and is obtained by setting $\theta_1^{E_2}$ and $\theta_2^{E_2}$ to 0.001. Based on these numerical results, the dynamics of the metabolic system can be characterized as follows.

- With fixed response thresholds, the system dynamics converge to certain fixed points, which are not altered by starting system evolution under different combined density vectors $\vec{X}$.
- For any configuration of the modeling parameters, the system dynamics possess at least one internal fixed point.
- Not all fixed points of the metabolic system are *stable* (or *attractive*). The stability of certain fixed points can be altered by changing agents' response thresholds.

Instead of analyzing the system behavior under fixed response thresholds, Figure 6.5 shows four different global dynamics when $\theta_1^{E_2}$ and $\theta_2^{E_2}$ change peri-

odically. Again, other response thresholds take values as in Table 6.2. In all the four scenarios, the system evolves initially with the same $\vec{X}$.

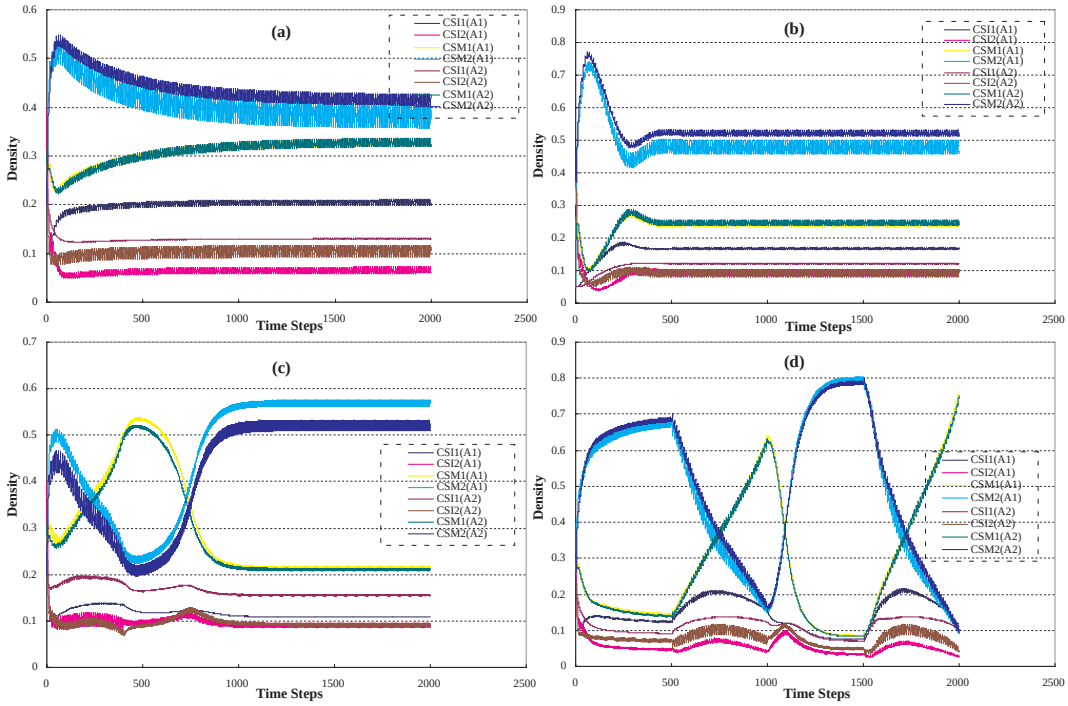

Figure 6.5: Four different system dynamics obtained by changing periodically $\theta_1^{E_2}$ and $\theta_2^{E_2}$.

Let $\pi_i^{E_j}$ denote the *periodic changing policy* of $Agent_i$ for threshold $\theta_i^{E_j}$. $\pi_i^{E_j}$ is a function of time such that, at any time step t, the value of $\theta_i^{E_j}$ equals to $\pi_i^{E_j}(t)$. $\pi_i^{E_j}$ is periodic with a *period* T in that $\pi_i^{E_j}(t) = \pi_i^{E_j}(t+T)$. Figures 6.5(a) and 6.5(b) are obtained when $\theta_1^{E_2}$ and $\theta_2^{E_2}$ follow fixed periodic changing policies. Figure 6.5(c) and 6.5(d) further indicate that much more complicated system dynamics can be exhibited by adopting varied periodic changing policies at different stages of the system evolution. They suggest that the metabolic system is able to satisfy a wide range of dynamical requirements by choosing properly each π^{E_j}.

6.4 Agent Coordination through Dynamic Optimization

The metabolic coordination approach is developed out of *Dynamic Optimization* algorithms (i.e. calculus of variations) [42]. It enables agents to systematically adjust (update) their response thresholds based on the environment feedback. The following distinguishing features can be identified as compared with conventional dynamic optimization approaches:

- The metabolic coordination approach enables agents to satisfy a system of inequality constraints, whereas only equality constraints are normally considered by dynamic optimization algorithms [42].

- In the metabolic approach, the optimization effort continues throughout the whole time period of the system evolution. Periodically, a separate optimization process is carried out to improve the system dynamics, and a memory-based updating mechanism is proposed to effectively utilize each $\pi_i^{E_j}$ identified previously. On the contrary, traditional dynamic optimization algorithms focus on systems that evolve from certain $\vec{X}$ for only a limited time period, and provide no guarantees regarding to long term system performance.

6.4.1 Agent Coordination through Dynamic Local Adjustment

Agent coordination in a metabolic system is achieved by requiring each agent to perform iteratively a local adjustment process. The major steps within each iteration are depicted in Figure 6.6, which cover totally $\tau_1 + \tau_2$ time steps. During this time period, each response threshold $\theta_i^{E_j}$ takes its value according to predetermined policies $\pi_i^{E_j}$. The policies adopted between consecutive iterations are adjusted by following sequentially four separate steps, namely, the exploration, evaluation, update, and extraction processes. The first three of them determine essentially the adjustment of an agent's response thresholds.

Repeat (for each iteration of the local adjustment process)

(1) Get changing policies π from the **policy respository**.

(2) **Explore** the system dynamics across $\tau_1 + \tau_2$ time steps.

(3) **Evaluate** $\Phi(\cdot)$ for the last τ_2 time steps.

(4) **Update** θ values among the first τ_1 time steps.

(5) **Extract** new changing policies from the modified response threshold values θ^*

(6) Store the identified policies into the policy repository.

End current iteration and Begin next iteration

Figure 6.6: Major steps of the local adjustment process.

The exploration process explores the global dynamics by evolving the system across $\tau_1 + \tau_2$ time steps. The system behavior during this time period is represented in the form of the following two lists:

The *state list:* $\left\{\vec{X}(t), \cdots, \vec{X}(t+\tau_1-1), \vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right\}$

and the *response threshold list:* $\left\{\vec{\Theta}(t), \cdots, \vec{\Theta}(t+\tau_1-1), \vec{\Theta}(t+\tau_1), \cdots, \vec{\Theta}(t+\tau_1+\tau_2-1)\right\}$

where $\vec{\Theta}(t)$ is a vector that combines all the local response thresholds of each agent at time step t. Each list is comprised of two parts, $Part_A$ and $Part_B$. $Part_A$ contains the first τ_1 elements of the respective list. It is used to adjust agents' response thresholds. $Part_B$ contains the rest τ_2 elements and is used for evaluating the system performance. The performance index J and the inequality constraints $g(\cdot)$ originally defined in Equation (6.6) are extended to another two functions $\varphi(\cdot)$ and $\Psi(\cdot)$ respectively. $\varphi(\cdot)$ is the *performance index* used by the local adjustment process and is defined as

$$\varphi\left(\vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right) = \frac{1}{\tau_2} \cdot \left(\sum_{i=0}^{\tau_2-1} \Phi\left(\vec{X}(t+\tau_1+i)\right)\right)$$

$\Psi(\cdot)$ represents the g inequality constraints and is constructed in exactly the same way as $\varphi(\cdot)$.

$$\Psi\left(\vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right) = \frac{1}{\tau_2} \cdot \left(\sum_{i=0}^{\tau_2-1} g\left(\vec{X}(t+\tau_1+i)\right)\right)$$

The local adjustment focuses primarily on the satisfaction of the g inequality constraints. When these constraints are satisfied, the performance index is further optimized. To highlight unsatisfied constraints, another function $\Psi^+(\cdot)$ is introduced, which is defined as

$$\begin{aligned}&\Psi^+\left(\vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right)\\ &= max\left(\Psi\left(\vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right), 0\right)\end{aligned}$$

Each entry of a vector returned by $\Psi^+(\cdot)$ takes the maximum between 0 and the corresponding entry of the vector returned by $\Psi(\cdot)$. As a result, any entry that equals to 0 indicates that the respective inequality constraint is satisfied. A *trimmed inequality function* $\hat{\Psi}(\cdot)$ is further defined as

$$\hat{\Psi}\left(\vec{X}(t+\tau_1), \cdots, \vec{X}(t+\tau_1+\tau_2-1)\right) = \left(\Psi^+_{k_1}(\cdot), \cdots, \Psi^+_{k_l}(\cdot)\right)^T$$

where $\Psi^+_{k_1} > 0, \cdots, \Psi^+_{k_l}(\cdot) > 0$. $\Psi^+_{k_i}$ denotes the k_i-th entry of the vector returned by $\Psi^+(\cdot)$.

The vector returned by $\hat{\Psi}(\cdot)$ only involves the positive entries of the vector returned by $\Psi^+(\cdot)$. In case when all entries of $\Psi^+(\cdot)$ are 0, $\hat{\Psi}(\cdot)$ returns 0 as well, indicating that all inequality constraints are satisfied. $\hat{\Psi}(\cdot)$ is *continuously differentiable* provided that $g(\cdot)$ holds the same property.

In order to describe the adjustment $Agent_i$ applies on its response thresholds between t and $t+\tau_1-1$, a vector $\vec{\theta_i}$ is introduced, which is of dimension $m \cdot \tau_1$,

$$\left(\theta_i^{E_1}(t), \cdots, \theta_i^{E_m}(t), \cdots, \theta_i^{E_1}(t+\tau_1-1), \cdots \theta_i^{E_m}(t+\tau_1-1)\right)^T$$

$\{E_1, \cdots, E_m\}$ is the set of enzymes that can be offered by $Agent_i$. $\theta_i^{E_j}(t_k)$ refers to the response threshold of $Agent_i$ for offering enzyme E_j at time step t_k. To ensure that the local adjustment is under the direction of improving the global system performance, the *local update process* (Ref. Figure 6.6) performed by $Agent_i$ during each iteration is to find a small change of the response thresholds, $\Delta\vec{\theta}_i$, as

$$\left(\Delta\theta_i^{E_1}(t), \cdots, \Delta\theta_i^{E_m}(t), \cdots, \Delta\theta_i^{E_1}(t+\tau_1-1), \cdots, \Delta\theta_i^{E_m}(t+\tau_1-1)\right)^T$$

such that

- $\Delta\vec{\theta}_i$ *optimizes* a linear approximation of the performance function $\varphi(\cdot)$ subject to a quadratic penalty on $\Delta\vec{\theta}_i$.
- $\Delta\vec{\theta}_i$ *satisfies* a group of equality constraints, which are in the form of linear approximations of the inequality constraints represented by $\hat{\Psi}(\cdot)$.

Taking into consideration both of the requirements, $\Delta\vec{\theta}_i$ leads the system dynamics towards a local optimum if one exists while at the same time satisfying the g inequality constraints. In order to identify such a $\Delta\vec{\theta}_i$, two matrices $\varphi_{\vec{\theta}_i}$ and $\hat{\Psi}_{\vec{\theta}_i}$ are required. $\varphi_{\vec{\theta}_i}$ measures the sensitivity of $\varphi(\cdot)$ with respect to $\vec{\theta}_i$. It has dimension $1 \times (m \cdot \tau_1)$ and is defined as

$$\left(\frac{\partial\varphi(\cdot)}{\partial\theta_i^{E_1}(t)}, \cdots, \frac{\partial\varphi(\cdot)}{\partial\theta_i^{E_m}(t)}, \cdots, \frac{\partial\varphi(\cdot)}{\partial\theta_i^{E_m}(t+\tau_1-1)}\right)$$

$\hat{\Psi}_{\vec{\theta}_i}$ quantifies the sensitivity of unsatisfied inequality constraints with regard to $\vec{\theta}_i$. It is of dimension $l \times (m \cdot \tau_1)$, where l is the number of unsatisfied inequality constraints, $l \leq g$.

$$\hat{\Psi}_{\vec{\theta}_i} = \begin{pmatrix} \frac{\partial\hat{\Psi}_1(\cdot)}{\partial\theta_i^{E_1}(t)} & \cdots & \frac{\partial\hat{\Psi}_1(\cdot)}{\partial\theta_i^{E_m}(t+\tau_1-1)} \\ \vdots & \ddots & \vdots \\ \frac{\partial\hat{\Psi}_l(\cdot)}{\partial\theta_i^{E_1}(t)} & \cdots & \frac{\partial\hat{\Psi}_l(\cdot)}{\partial\theta_i^{E_m}(t+\tau_1-1)} \end{pmatrix}$$

$\hat{\Psi}_i(\cdot)$ denotes the i-th entry of the vector returned by $\hat{\Psi}(\cdot)$. With the two matrices, $\Delta\vec{\theta}_i$ is regarded as a solution of an optimization problem, which is to minimize a function $\Delta\varphi(\cdot)$ such that

$$min\left(\Delta\varphi(\cdot)\right) = \varphi_{\vec{\theta}_i} \times \Delta\vec{\theta}_i + \frac{\Delta\vec{\theta}_i^T \times \Delta\vec{\theta}_i}{2 \cdot k} \tag{6.8}$$

subject to a group of equality constraints,

$$\hat{\Psi}_{\vec{\theta}_i} \times \Delta\vec{\theta}_i = -\eta \cdot \hat{\Psi}(\cdot) \tag{6.9}$$

where k and η are two positive real numbers. If $\hat{\Psi}(\cdot)$ is linear with respect to $\vec{\theta}_i$, one step of adjustment by $\Delta\vec{\theta}_i$ will make the system dynamics satisfy all the currently unsatisfied inequality constraints, provided that the metabolic system starts evolving from $\vec{X}(t)$. In other words, by choosing a proper η (e.g. $\eta \geq 1$), $\Delta\vec{\theta}_i$, which satisfies the above equality constraints, will lead the system dynamics towards the satisfaction of the g inequality constraints.

To solve the optimization problem as defined in Equation (6.8) and (6.9), a *Lagrangian function* is formed by adjoining the constraint part with a *Lagrange multiplier vector* $\vec{\lambda}$.

$$\Delta J = \varphi_{\vec{\theta}_i} \times \Delta\vec{\theta}_i + \frac{\Delta\vec{\theta}_i^T \times \Delta\vec{\theta}_i}{2 \cdot k} + \vec{\lambda}^T \times \left(\hat{\Psi}_{\vec{\theta}_i} \times \Delta\vec{\theta}_i + \eta \cdot \hat{\Psi}(\cdot)\right) \tag{6.10}$$

Take the differential of ΔJ with respect to $\Delta\vec{\theta}_i$ and assume that $d(\Delta J) = 0$,

$$\varphi_{\vec{\theta}_i} + \frac{\Delta\vec{\theta}_i^T}{k} + \vec{\lambda}^T \times \hat{\Psi}_{\vec{\theta}_i} = 0 \tag{6.11}$$

Solve Equation (6.11) with regard to $\Delta\vec{\theta}_i$

$$\Delta\vec{\theta}_i^T = -k \cdot \left(\varphi_{\vec{\theta}_i} + \vec{\lambda}^T \cdot \hat{\Psi}_{\vec{\theta}_i}\right) \tag{6.12}$$

Substitute Equation (6.12) into Equation (6.9),

$$-k \cdot \hat{\Psi}_{\vec{\theta}_i} \times \varphi_{\vec{\theta}_i}^T - k \cdot \hat{\Psi}_{\vec{\theta}_i} \times \hat{\Psi}_{\vec{\theta}_i}^T \times \vec{\lambda}^T + \eta \cdot \hat{\Psi} = 0 \tag{6.13}$$

Solve this equation concerning $\vec{\lambda}$

$$\vec{\lambda} = -Q^{-1} \times \hat{\Psi}_{\vec{\theta}_i} \times \varphi_{\vec{\theta}_i} + \frac{\eta}{k} \cdot Q^{-1} \times \hat{\Psi} \tag{6.14}$$

where $Q = \hat{\Psi}_{\vec{\theta}_i} \times \hat{\Psi}_{\vec{\theta}_i}^T$. Substituting Equation (6.12) into (6.14), the desired local adjustment $\Delta\vec{\theta}_i$ is obtained as

$$\Delta\vec{\theta}_i = -\eta \cdot \hat{\Psi}_{\vec{\theta}_i}^* \times \hat{\Psi} - k \cdot H_{\vec{\theta}_i}^T \tag{6.15}$$

where $\hat{\Psi}_{\vec{\theta}_i}^* = \hat{\Psi}_{\vec{\theta}_i} \times Q^{-1}$ and $H_{\vec{\theta}_i} = \varphi_{\vec{\theta}_i} + \vec{\lambda}^T \times \hat{\Psi}_{\vec{\theta}_i}$

To focus on the satisfaction of the g inequality constraints, k in Equation (6.15) approaches to 0 when many constraints are significantly violated, that is, the modulus of the vector returned by $\hat{\Psi}(\cdot)$, $\left\|\hat{\Psi}(\cdot)\right\|$, exceeds certain positive number δ. On the other hand, when all inequality constraints are satisfied or at the brink of satisfaction, k is increased to a larger value in order to optimize the

performance index J. After determining $\Delta\vec{\theta}_i$, the response thresholds taken by $Agent_i$ between time steps t and $t+\tau_1-1$ are updated accordingly to obtain

$$\vec{\theta}_i^* = \vec{\theta}_i + \Delta\vec{\theta}_i$$

Although $\vec{\theta}_i$ is derived from periodic changing policies, the updated response thresholds $\vec{\theta}_i^*$ may not be periodic any more. In order to identify a new policy $\pi_i^{E_j}$ for each $\theta_i^{E_j}$ based on $\vec{\theta}_i^*$, the policy extraction process is performed right after the update process. For a vector of threshold values between t and $t+\tau_1-1$,

$$\vec{\Theta}_i^{E_j} = \left(\theta_i^{E_j}(t), \theta_i^{E_j}(t+1), \cdots, \theta_i^{E_j}(t+\tau_1-1)\right)^T$$

The objective of the policy extraction process is to find a periodic policy $\pi_i^{E_j}$ with period T, such that the *distance* between $\vec{\Theta}_i^{E_j}$ and the vector

$$\overrightarrow{\Theta}_i^{E_j} = \left(\pi_i^{E_j}(t), \pi_i^{E_j}(t+1), \cdots, \pi_i^{E_j}(t+\tau_1-1)\right)^T$$

is minimized. The distance between $\vec{\Theta}_i^{E_j}$ and $\overrightarrow{\Theta}_i^{E_j}$ is defined as

$$\left\langle \vec{\Theta}_i^{E_j}, \overrightarrow{\Theta}_i^{E_j} \right\rangle = \sqrt{\sum_{k=0}^{\tau_1-1} \left(\theta_i^{E_j}(t+k) - \pi_i^{E_j}(t+k)\right)^2} \tag{6.16}$$

To restrict the range of policies considered, an upper bound on the acceptable period of any policies, T^{Upp}, is imposed. All policies identified by the policy extraction process will have a period $T \leq T^{Upp}$. The acceptable policies that minimize Equation (6.16) can be identified easily and are considered as the best approximation of $\vec{\theta}_i^*$. Clearly, when $T^{Upp} = \tau_1$, the policy extraction process has no effects on the threshold values $\vec{\theta}_i^*$ obtained previously. The extraction process is used for two main reasons:

- Representing policies periodically can save system memory. Instead of remembering threshold values at each time step between t and $t+\tau_1-1$, the threshold values within a single period will be stored. This is especially useful when shop floor agents have limited memory space.

- By averaging out the difference of $\theta_i^{E_j}$ at separate time steps, the policy extraction process helps filter out inappropriate adjustment ($\Delta\theta_i^{E_j}$) due to numerical calculation errors or inaccurate modeling parameters. When system dynamics are far-from satisfaction, $\vec{\theta}_i^*$ may change abruptly between consecutive iterations (Ref. Figure 6.6). As a result, use $\vec{\theta}_i^*$ directly may result in rather poor system performance. Therefore, a conservative approach using the policy extraction process is taken in our approach to ensure that the performance will not become too low as the system evolves.

For each $\theta_i^{E_j}$ of $Agent_i$, the policy extraction process produces a policy $\pi_i^{E_j}$ to be stored in $Agent_i$'s *policy repository* (Ref. Figure 6.6). Any $\pi_i^{E_j}$ in the repository serves two purposes. First, if the combined density vector $\vec{X}(t+\tau_1+\tau_2-1)$ has never been encountered previously, $\pi_i^{E_j}$ will be directly used by $Agent_i$ as the policy of $\theta_i^{E_j}$ during the next iteration. Second, when the combined density vector $\vec{X}(t)$ is re-encountered at a future iteration ($Iteraction_A$), $\pi_i^{E_j}$ will be utilized to identify a new policy to be used during that iteration. Suppose that the policy of $\theta_i^{E_j}$ identified before $Iteraction_A$ is $\overline{\pi}_i^{E_j}$, then the policy to be used during $Iteration_A$ is

$$\overline{\overline{\pi}}_i^{E_j}(t) = (1-\alpha)\cdot\overline{\pi}_i^{E_j}(t) + \alpha\cdot\pi_i^{E_j}(t)$$

where α is nonnegative and less than 1.

6.4.2 Numerical Evaluation with a Simple Metabolic System

The metabolic system introduced in subsection 6.3.2 is used to evaluate the effectiveness of the local adjustment approach. In order to reduce the computational complexity, only response thresholds related to enzyme E_2 ($\theta_1^{E_2}$ and $\theta_2^{E_2}$) will be adjusted locally by the two agents, $Agent_1$ and $Agent_2$. Other response thresholds are all set to 1. In the first experiment, the performance index J is kept constant in order to examine whether the local adjustment approach can lead the system dynamics towards the satisfaction of 8 inequality constraints, which are

$$0.3 \le Den^1(CSM_1) \le 0.4,\ 0.3 \le Den^1(CSM_2) \le 0.4,$$
$$0.3 \le Den^2(CSM_1) \le 0.4,\ 0.3 \le Den^2(CSM_2) \le 0.4.$$

Each of the 4 inequalities represents two inequality constraints. In the experiment, every iteration of the local adjustment process spans 23 time steps, among which the number of time steps used for adjustment (τ_1) equals to 20, and the number of time steps used for system evaluation (τ_2) equals to 3. As shown in Figure 6.7, the global system dynamics are able to converge towards certain restricted region which satisfies the 8 inequality constraints, provided that T^{Upp} is reasonably small. This is the case when T^{Upp} equals to 1, 2, 3, 4, or 5. When T^{Upp} keeps increasing and reaches 10, the local adjustment becomes unstable and the given inequality constraints cannot be satisfied anymore. Figure 6.8 further shows that, when $T^{Upp}=3$, the metabolic coordination approach enjoys good stability and is effective with regard to arbitrary $\vec{X}$ from which the system evolves.

The second experiment indicates that with different τ_2, the metabolic system is able to exhibit varied dynamics as well. In the experiment, $\tau_1=20$, $T^{Upp}=5$. Figure 6.9 shows that as τ_2 keeps increasing, the system dynamics become more

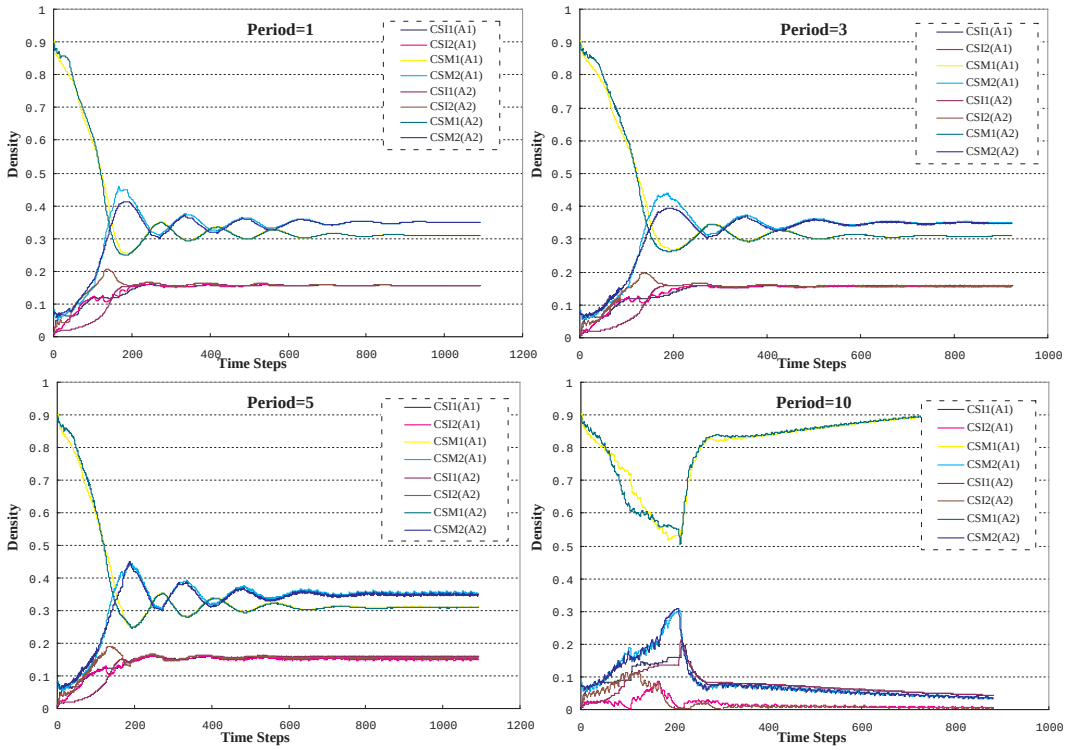

Figure 6.7: The dynamics of the coordination approach under varied T^{Upp}.

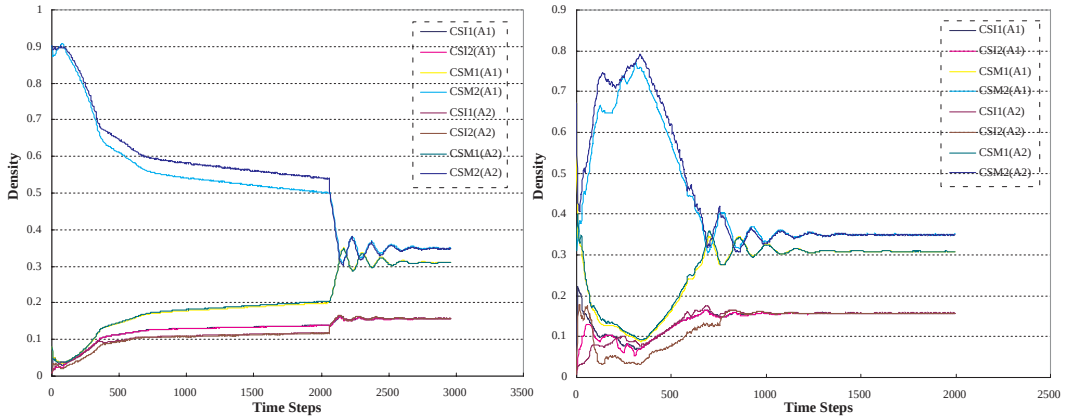

Figure 6.8: The global system dynamics when $T^{Upp} = 3$.

and more vibrant. When τ_2 reaches 20, the densities of CSM_1 and CSM_2 of the two agents fail to converge to any restricted regions where the 8 inequality constraints (the same as used in the first experiment) can be satisfied. To respect this observation, τ_2 is set to 1 in the shop floor system to be described in the next section.

The effectiveness of the *memory-based* updating mechanism is evaluated in the third experiment, where agents utilize their *policy repositories* as stated in Figure 6.6. The 8 inequality constraints used by this experiment are

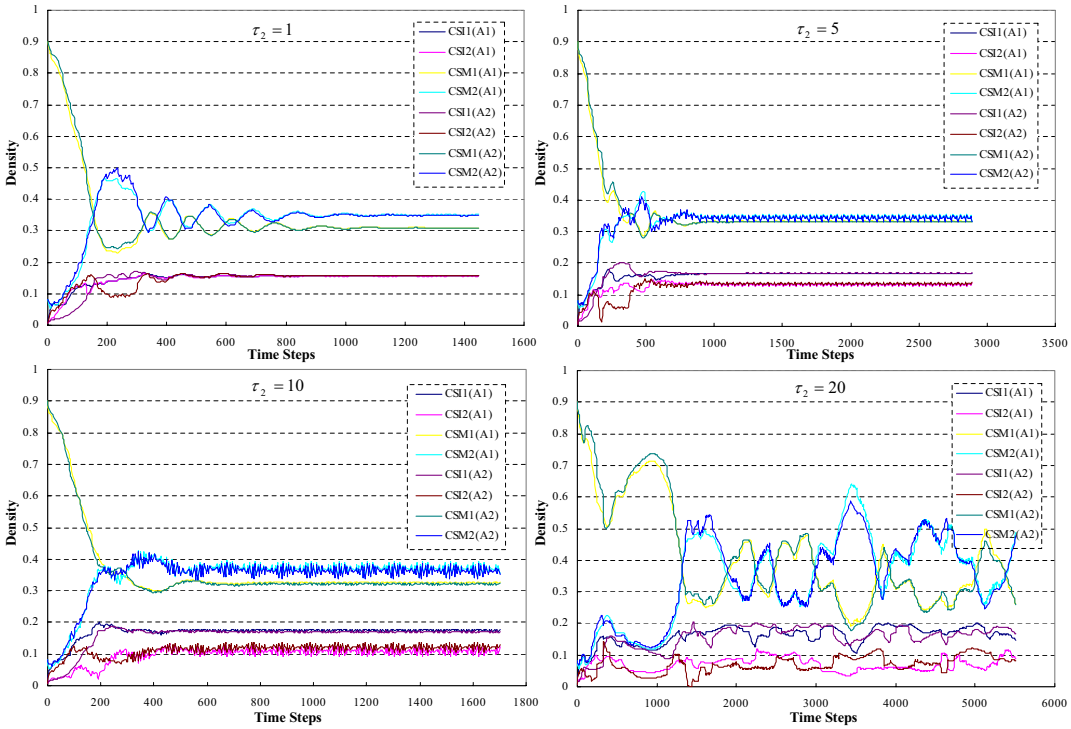

Figure 6.9: The global system dynamics under varied τ_2.

$$0.2 \leq Den^1(CSM_1) \leq 0.3,\ 0.5 \leq Den^1(CSM_2) \leq 0.6,$$
$$0.2 \leq Den^2(CSM_1) \leq 0.3,\ 0.5 \leq Den^2(CSM_2) \leq 0.6.$$

As policies stored in the policy repository have already undergone a series of local adjustments, they are more likely to satisfy the given inequality constraints. As a result, when the system starts evolving from similar $\vec{X}$, the system dynamics might be different and should converge more quickly as the repetition continues. This fact can be seen from the four different system dynamics shown in Figure 6.10.

All experiment results depicted in Figure 6.10 are obtained when $T^{Upp} = 5$, $\tau_2 = 5$, and $\alpha = 0.3$. As shown in Figure 6.10, when the system starts evolving from the same $\vec{X}$ for 10 times (the 10-th repetition), it takes only 1 overshoot (see the density changes of CSM_2 of the two agents) for the metabolic system to converge. In comparison, the system experiences 5 ups and downs before convergence during the first repetition (i.e. when the policy repository is empty initially).

The last experiment is performed with 8 inequality constraints, which are

$$0.2 \leq Den^1(CSM_1) \leq 0.3,\ 0.4 \leq Den^1(CSM_2) \leq 0.7,$$
$$0.2 \leq Den^2(CSM_1) \leq 0.3,\ 0.4 \leq Den^2(CSM_2) \leq 0.7,$$

and a quadratic performance index J,

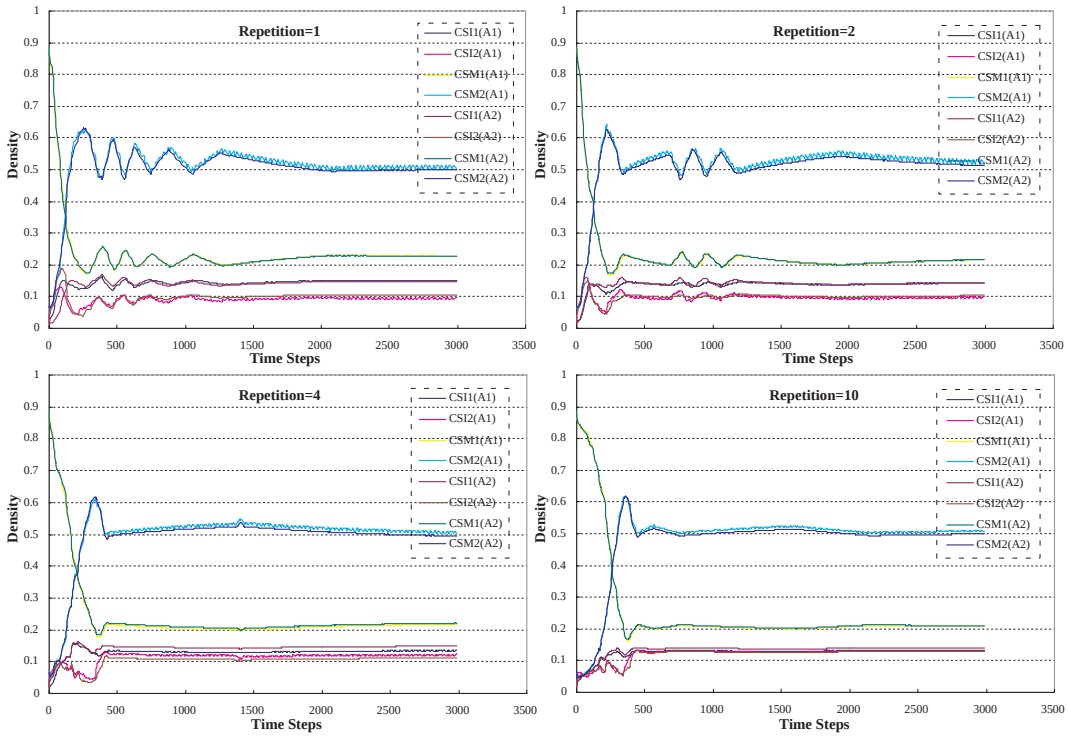

Figure 6.10: The global system dynamics during different repetitions.

$$J = \Big(\; Den^1(CSM_1) + Den^1(CSM_2) + Den^2(CSM_1) + Den^2(CSM_2) \; \Big)^2$$

The experiment follows the same settings as the third experiment. The results show that the coordination approach effectively coordinates the two agents such that no inequality constraints are violated and the performance index J is minimized as far as possible. Specifically, Figure 6.11 depicts the system dynamics and the change of J obtained after the metabolic system has evolved from the same combined density vector for three times.

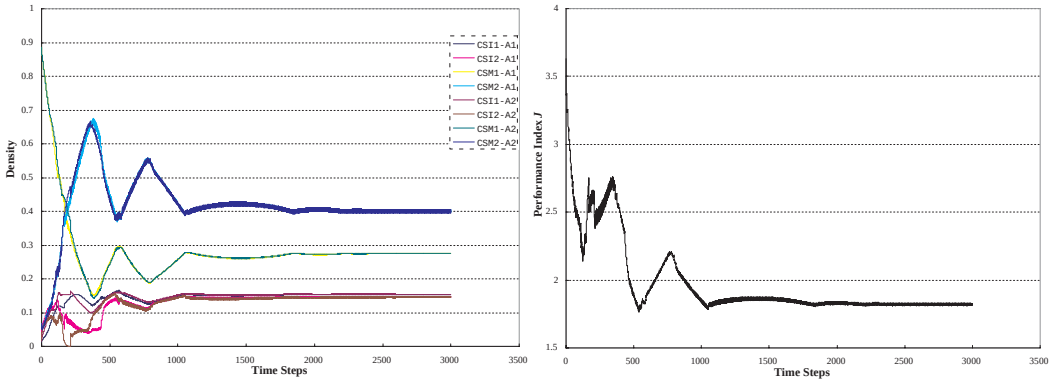

Figure 6.11: The global system dynamics and the change of the performance index J.

The system dynamics shown at the left part of Figure 6.11 indicates that after two major ups and downs, the system finally converges to certain restricted region where all inequality constraints are satisfied. When the system converges, the performance index J reaches and stays at a minimum value as shown at the right part of Figure 6.11.

6.5 Application: a Multi-Agent Shop Floor Manufacturing System

In this Section, the metabolic coordination approach is applied to a multi-agent shop floor manufacturing system. The high-level structure of the system is shown in Figure 6.12. In Figure 6.12, each circle represents a group of 5 identical manufacturing devices. Each device is controlled by a separate agent. A total of 30 agents are thus employed by the shop floor to produce three different types of products: CSO_1, CSO_2 and CSO_3. These products are manufactured from four types of parts provided to the shop floor, namely, CSI_1, CSI_2, CSI_3, and CSI_4. In order to manufacture a product, the parts are transformed intermediately into partially manufactured products, which are transferred between agents in separate groups. For example, CSM_3 is produced by agents in Group1 and is further transferred to agents of Group3 (Ref. Figure 6.12). The procedure to manufacture these products is represented as a metabolic network shown in Figure 6.13.

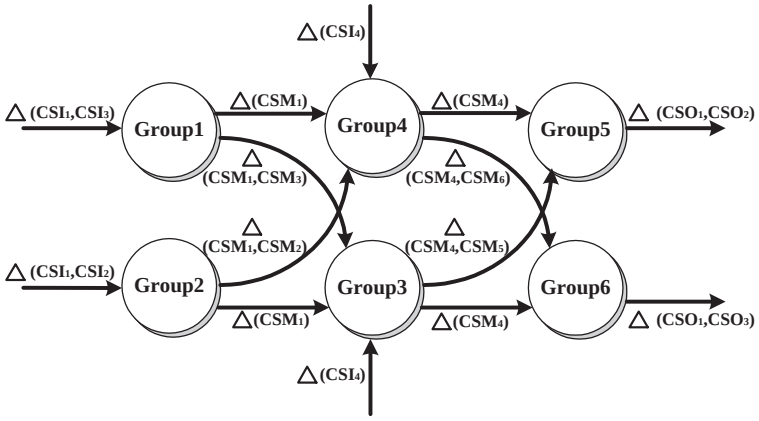

Figure 6.12: High-level structure of a multi-agent manufacturing system.

The enzymes that link the various parts in Figure 6.13 stand for the actual manufacturing operations to be performed. As any single agent is incapable of offering all types of enzymes, the manufacturing of any product requires the joint effort of multiple agents. The grouping of agents and the enzymes offered by each group are listed in Table 6.3. There is also a designated *environment agent* responsible for providing unprocessed parts and collecting final products.

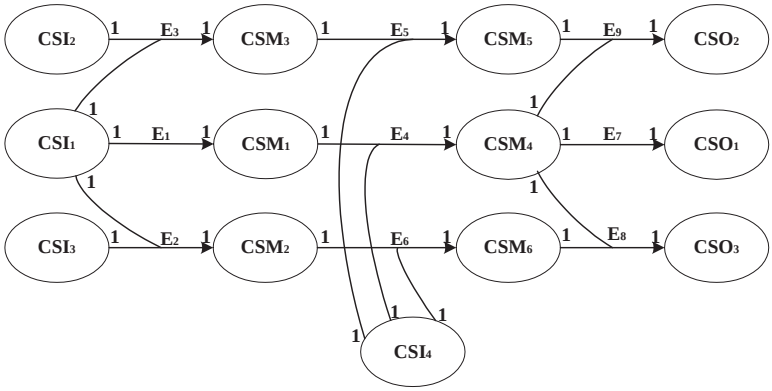

Figure 6.13: The metabolic network of our manufacturing system.

Group	Group1	Group2	Group3	Group4	Group5	Group6
Agents	$Agent_1$ – $Agent_5$	$Agent_6$ – $Agent_{10}$	$Agent_{11}$ – $Agent_{15}$	$Agent_{16}$ – $Agent_{20}$	$Agent_{21}$ – $Agent_{25}$	$Agent_{26}$ – $Agent_{30}$
Enzymes	E_1,E_3	E_1,E_2	E_4,E_5	E_4,E_6	E_7,E_9	E_7,E_8
Parts Contained	CSI_1, CSI_3, CSM_1, CSM_3	CSI_1, CSI_2, CSM_1, CSM_2	CSM_1, CSM_3, CSM_4, CSM_5, CSI_4	CSM_1, CSM_2, CSM_4, CSM_6, CSI_4	CSM_4, CSM_5, CSO_1, CSO_2	CSM_4, CSM_6, CSO_1, CSO_3
Response Thresholds	$\theta_i^{E_1}$, $\theta_i^{E_3}$	$\theta_i^{E_1}$, $\theta_i^{E_2}$	$\theta_i^{E_4}$, $\theta_i^{E_5}$	$\theta_i^{E_4}$, $\theta_i^{E_6}$	$\theta_i^{E_7}$, $\theta_i^{E_9}$	$\theta_i^{E_7}$, $\theta_i^{E_8}$

Table 6.3: The IDs of, the enzymes offered by, and the parts contained in agents of each group.

At each time step, the environment agent provide four types of parts (i.e. CSI_1, CSI_2, CSI_3, and CSI_4) to agents belong to from Group1 to Group4 (Ref. Figure 6.12). It also collects from agents belong to Group5 and Group6 the final products manufactured (i.e. CSO_1, CSO_2 and CSO_3). The performance of this shop floor system is measured as the amount of products received by the environment agent during fixed time period. The following are the particular settings of our experiments.

- All system behaviors are carried out according to the increasing time steps, which serve as the standard measure of time.
- Each manufacturing operation consumes 2 time steps to complete[3].
- An agent can only perform one operation at a time.
- Each agent uses its *part pool* to store up to 25 parts, which have been processed or are still under processing.

[3]Our coordination approach imposes no restrictions on the required time steps of performing any manufacturing operations. The same number of time steps is used in the experiment in order to ease the calculation of the system efficiency, as evidenced later in this Section.

- Agents communicate with each other by sending messages. The number of messages sent by any agent at any time step is not restricted.

- The transportation of parts or products takes one time step to complete. Each agent (except the environment agent) is only capable of transferring one part at a time.

In the remaining part of this section, agents' dynamic behaviors and the corresponding system performance are examined first. All shop floor agents are constructed based upon the metabolic system model introduced in Section 6.3. In order to use the coordination approach proposed in Section 6.4, an additional step, which is termed the *model identification*, is to be performed by each agent. The details of this step are described right after the analysis of the system dynamics when no local adjustment is applied.

As described in Section 6.3, each shop floor agent performs three processes (i.e. absorption, transformation, and release) at each time step. The transformation process involves the interaction of three components: the *part pool*, the *transformation unit*, and the *enzyme inspiration unit*. The part pool provides three operations directly accessible by the other two components: (1) *Take part*: the operation to take a part from the pool; (2) *Store part*: the operation to put a part into the pool; and (3) *Part density*: the operation to obtain the densities (amount) of parts contained in the pool. The first two operations are used by the transformation unit. The main procedure is as follows. The transformation unit takes all parts to be transformed by a given enzyme through calling the *take part* operation. After finishing the corresponding manufacturing operation, the newly created parts are stored back to the part pool by calling the *store part* operation. This procedure alters part densities, which are obtained by calling the *part density* operation and is used by the enzyme inspiration unit to decide which enzyme should be offered to the transformation unit.

The enzyme inspiration unit examines whether the agent is busy performing a manufacturing operation. If yes, no enzymes will be offered. Otherwise, the unit offers randomly an enzyme E_j with a probability $Pr(E_j)$. $Pr(E_j)$ is evaluated from Equations (6.2) to (6.5).

In order to avoid manufacturing too many parts of the same type, for all the experiments, the densities of any parts contained in the part pool are upper bounded and cannot exceed 0.2. For example, $Agent_1$ is only allowed to hold at most $0.2 \times 25 = 5$ CSM_1. As a result, if $Agent_1$ already holds 5 or more CSM_1 in its part pool, it will not perform any manufacturing operation represented by enzyme E_1 to produce more CSM_1.

The release and absorption processes are performed by each agent at every time

step. Only those parts as indicated in Figure 6.12 are allowed to be transferred between agents in separate groups. We use the *density diffusion* mechanism given in Equation (6.1) to decide which parts will be transferred. For example, the probabilities for $Agent_1$ to release CSM_1 and CSM_3 are

$$k_{re}^{CSM_1}\frac{Den^1(CSM_1)}{Den^1(CSM_1)+Den^1(CSM_3)} \text{ and } k_{re}^{CSM_3}\frac{Den^1(CSM_3)}{Den^1(CSM_1)+Den^1(CSM_3)}$$

respectively. $Den^i(Part_j)$ gives the density of $Part_j$ contained in $Agent_i$. The part chosen may not be transferred to other agents. At each time step, the density of any parts after the absorption process should not exceed 0.2. In order to determine whether or not a part can be transferred, an agent, such as $Agent_1$, will interact with other agents by following a predefined protocol, similar to the contract net protocol [216, 255].

The protocol works as follows. Use CSM_1 as an example. $Agent_1$ knows that 10 agents ($Agent_{11}$ to $Agent_{20}$) require CSM_1. It asks these 10 agents whether it can transfer a CSM_1 to them. Upon receiving this message, the 10 agents check respectively their local densities of CSM_1. If the density value is above 0.2, they will reply $Agent_1$ with a denial indication. Otherwise, a reply message with an agreement indication is sent to $Agent_1$. After all reply messages have been received, $Agent_1$ examines whether there are agents waiting for CSM_1. If the answer is negative, CSM_1 will not be released. Otherwise, $Agent_1$ selects randomly one agent that needs CSM_1, such as $Agent_{11}$, and sends an acknowledgement message to it. $Agent_{11}$ in turn examines whether or not it can store this extra CSM_1. If yes, it reserves the place in its part pool and sends a confirmation message back. If not, a denial message will be received by $Agent_1$. After receiving the reply, $Agent_1$ makes its final decision on either transferring CSM_1 to $Agent_{11}$ (in case of an agreement indication) or not (in case of a denial indication).

With agents' behaviors being described as above, the global dynamics of this shop floor system are explored when agents take predefined policies. In the experiment, every agent uses the same two policies for its two response thresholds (Ref. Table 6.3). One period of each policy is given by the lists:

$$List_1 = \{0.1, 0.5, 0.15\} \text{ and } List_2 = \{0.1, 0.8, 0.15\}$$

The whole time period during which the shop floor system evolves is divided into consecutive *sessions* (Ω). Each session spans 10 time steps. Within every session, agents' response thresholds stay at a value in either of the two lists. As the session increases, the thresholds take accordingly the adjacent values in the lists. Figure 6.14 depicts the dynamic density change of six agents, each of which belongs to a separate group. Some data lines in the Figure are indexed by the legend "*Free*" that refers to the *free spaces* in these agents' part pools.

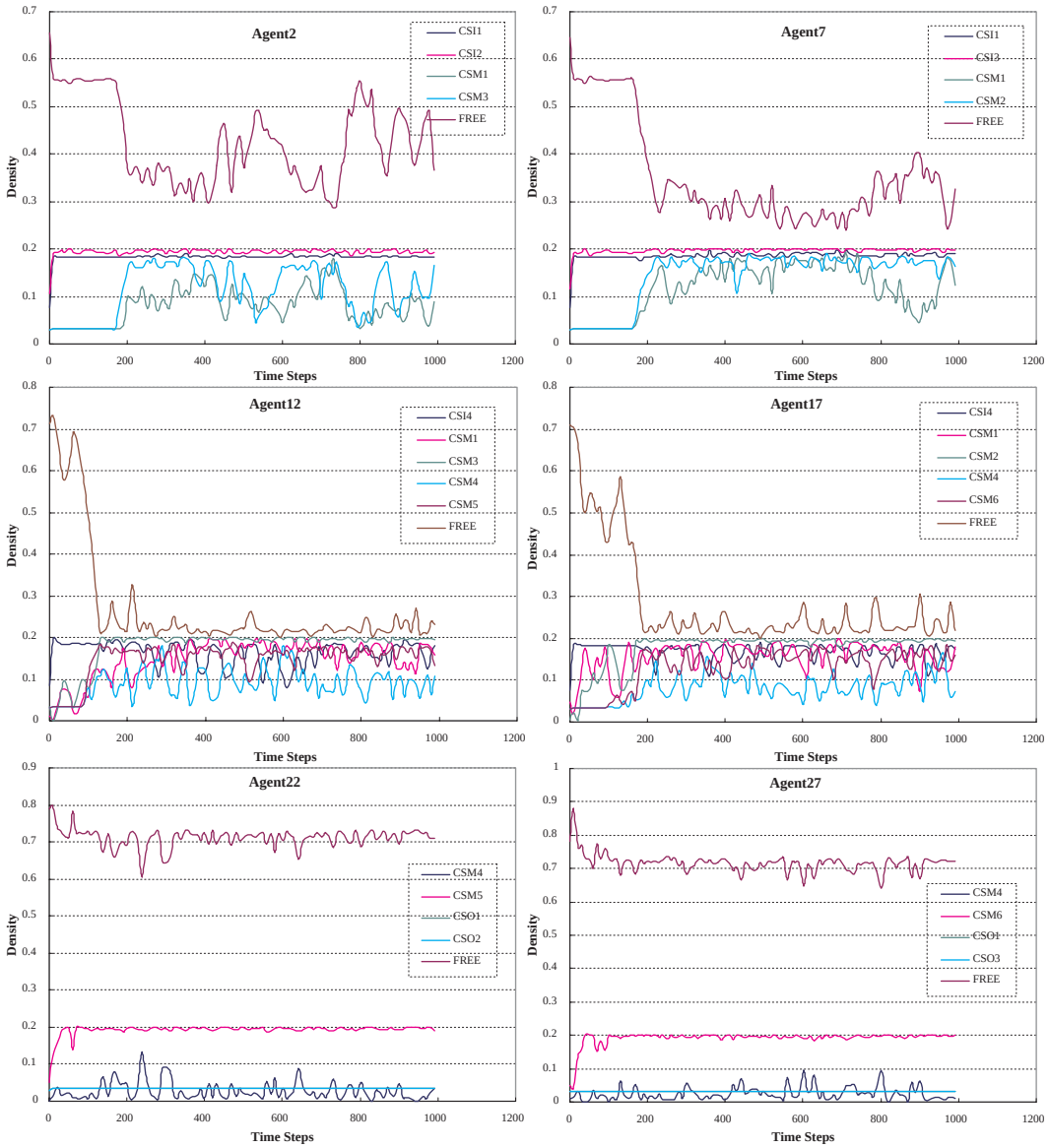

Figure 6.14: The dynamics of our manufacturing agents that take predefined policies for each of the response thresholds they have.

The amount of final products manufactured during each session is collected by the environment agent and shown at the left part of Figure 6.15. The *average* amount of products manufactured and the *average system efficiency* within each session between time steps 60 and 1000 are also listed in Table 6.4.

CSO_1	CSO_2	CSO_3	Efficiency
16.563	6.521	6.254	0.842

Table 6.4: The average amount of products manufactured and the average system efficiency.

Table 6.4 indicates that CSO_1 is manufactured at more than twice the speed

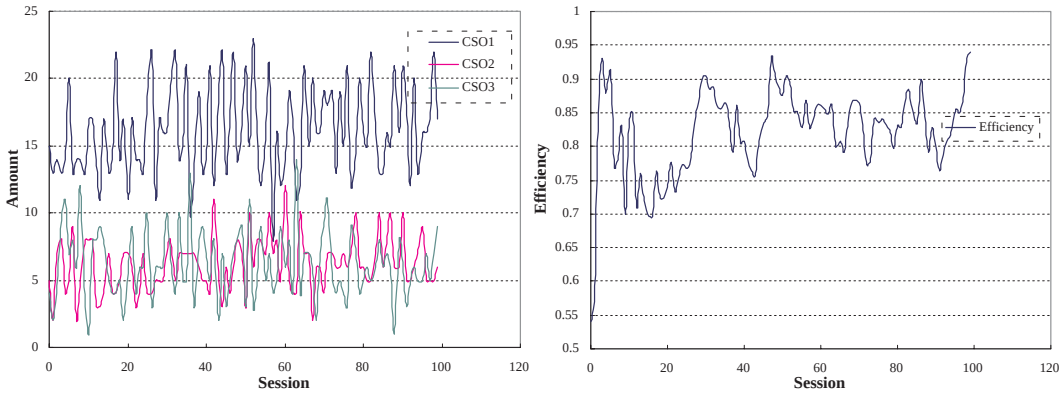

Figure 6.15: The amount of products manufactured and the system efficiency during each session between time steps 1 and 1000.

of manufacturing CSO_2 and CSO_3. The average system efficiency is only at 0.842, which is expected to be improved in order to increase the system throughput. The *efficiency* measures the proportion of manufacturing capabilities that have been actually utilized to manufacture products. In the shop floor system, producing one CSO_2 or one CSO_3 requires twice the effort of producing one CSO_1. For example, manufacturing one CSO_2 requires agents in Group3 and Group4 to produce both one CSM_4 and one CSM_5. Since Group3 and Group4 involve totally 10 agents and every manufacturing operation takes 2 time steps to complete, during any session Ω_i of 10 time steps, the *efficiency* of the manufacturing system is evaluated as

$$\frac{N(CSO_1, \Omega_i) + 2 \cdot N(CSO_2, \Omega_i) + 2 \cdot N(CSO_3, \Omega_i)}{(10 \times 10)/2} \tag{6.17}$$

where $N(CSO_j, \Omega_i)$ gives the amount of CSO_j manufactured within session Ω_i. In order to improve the system efficiency, each agent adopts the metabolic coordination approach proposed in this Chapter to solve a dynamic optimization problem as defined in Equation (6.6). The performance index J to be minimized is exactly the negation of Equation (6.17). Three inequality constraints are used to regulate further that all types of products should be manufactured at the same speed. They are

$$N(CSO_1, \Omega_i) \leq N(CSO_2, \Omega_i),\ N(CSO_2, \Omega_i) \leq N(CSO_3, \Omega_i),\ \text{and}$$
$$N(CSO_3, \Omega_i) \leq N(CSO_1, \Omega_i)$$

To carry out the local adjustment, agents need to go through an extra *model identification* step before the update process (Ref. Figure 6.6). Their dynamic models establish the mathematical relation between *average* part densities of consecutive sessions. The *average density vector* of $Agent_i$ in session Ω_j (starting at time step t) is

$$\vec{\overline{x}}_i(\Omega_j) = (1/10) \cdot \sum_{k=0}^{9} \vec{x}_i(t+k)$$

For all shop floor agents, the main objective of the *model identification* step is to estimate four modeling parameters. Take $Agent_{11}$ as an example, the four parameters to be identified during each session by $Agent_{11}$ are k^{E_4}, k^{E_5}, $k_{re}^{CSM_4}$ and $k_{re}^{CSM_5}$. k^{E_4} and $k_{11}^{E_5}$ are *reaction factors* that measure the efficiency of the transformation process. The other two are *release ratios* that govern $Agent_{11}$'s capability of transferring parts to other agents. Since parts absorbed from other agents during past sessions have already been known, there is no need to include the release ratios of other agents into $Agent_{11}$'s dynamic model. In the experiment, the model identification step performed by each agent is to find the four parameters that best approximate the density changes across the last 5 sessions. Numerous system identification techniques are provided in the literature to efficiently solve this problem [185]. As our major concern is the metabolic coordination approach, the details will be omitted.

Given the dynamic model identified, an agent is able to evaluate the change of the amount of parts it produces with regard to the change of its local response thresholds and part densities. They are used for calculating the two matrices, $\varphi_{\vec{\theta}_i}$ and $\hat{\Psi}_{\vec{\theta}_i}$, which are essential to the update process (Ref. Figure 6.6). Since the global performance index and inequality constraints are all defined over the amount of final products manufactured, the calculation of the two matrices is performed backwardly from agents in Group5 and Group6, to agents in Group3 and Group4, and finally to agents in Group1 and Group2. The calculation does not require all agents to broadcast their local information within the whole agent community. The basic procedure is sketched as follows.

At the last time step of each session, agents belong to Group5 and Group6 first obtain from the environment agent the total amount of final products manufactured during that session. This information will be applied to evaluate the system efficiency and examine the satisfaction of the 3 inequality constraints. Based on the evaluation result, each agent in Group5 and Group6 calculates locally their matrices $\varphi_{\vec{\theta}_i}$ and $\hat{\Psi}_{\vec{\theta}_i}$. Afterwards, they broadcast their dynamic models and the manufacturing information to all agents in Group3 and Group4. Upon receiving the broadcasted messages, these agents in turn calculate their own $\varphi_{\vec{\theta}_i}$ and $\hat{\Psi}_{\vec{\theta}_i}$, and broadcast their dynamic models and manufacturing information to agents in Group1 and Group2. Finally, agents in Group1 and Group2 integrate all the dynamics models they received and calculate accordingly their $\varphi_{\vec{\theta}_i}$ and $\hat{\Psi}_{\vec{\theta}_i}$.

In our implementation of the local adjustment process, the time period τ_1 covers the past 5 sessions, while the time period τ_2 involves only the current

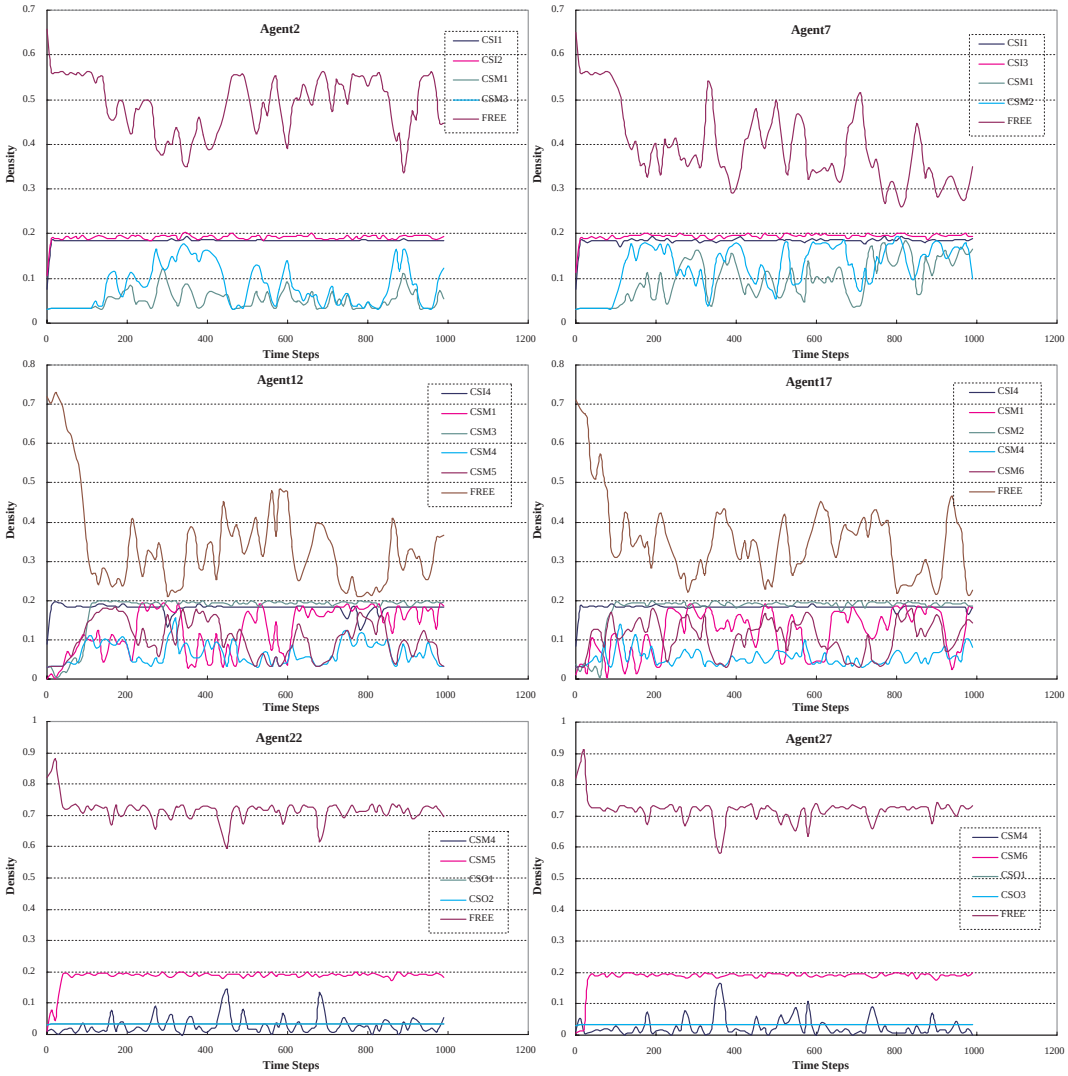

Figure 6.16: The dynamics of the manufacturing agents after using the metabolic coordination approach.

CSO_1	CSO_2	CSO_3	Efficiency
9.4789	9.6197	9.2394	0.944

Table 6.5: The average amount of products manufactured and the average system efficiency when the proposed coordination approach is applied.

session. The upper bound on the allowable period of any policies, T^{Upp}, is 3. Each shop floor agent performs the local adjustment process at the last time step of every session. η is set to 0.8. The variable k used for evaluating $\Delta\vec{\theta}$ (Ref. Equation (6.15)) takes its value according to the satisfaction of the 3 inequality constraints. When the difference among $N(CSO_1, \Omega_i)$, $N(CSO_2, \Omega_i)$, and $N(CSO_3, \Omega_i)$ does not exceed 0.5, k takes as its value 0.8. If the difference is between 0.8 and 1.2, $k = 0.5$. Finally, if the difference is above 1.2, the value of k decreases to 0.1.

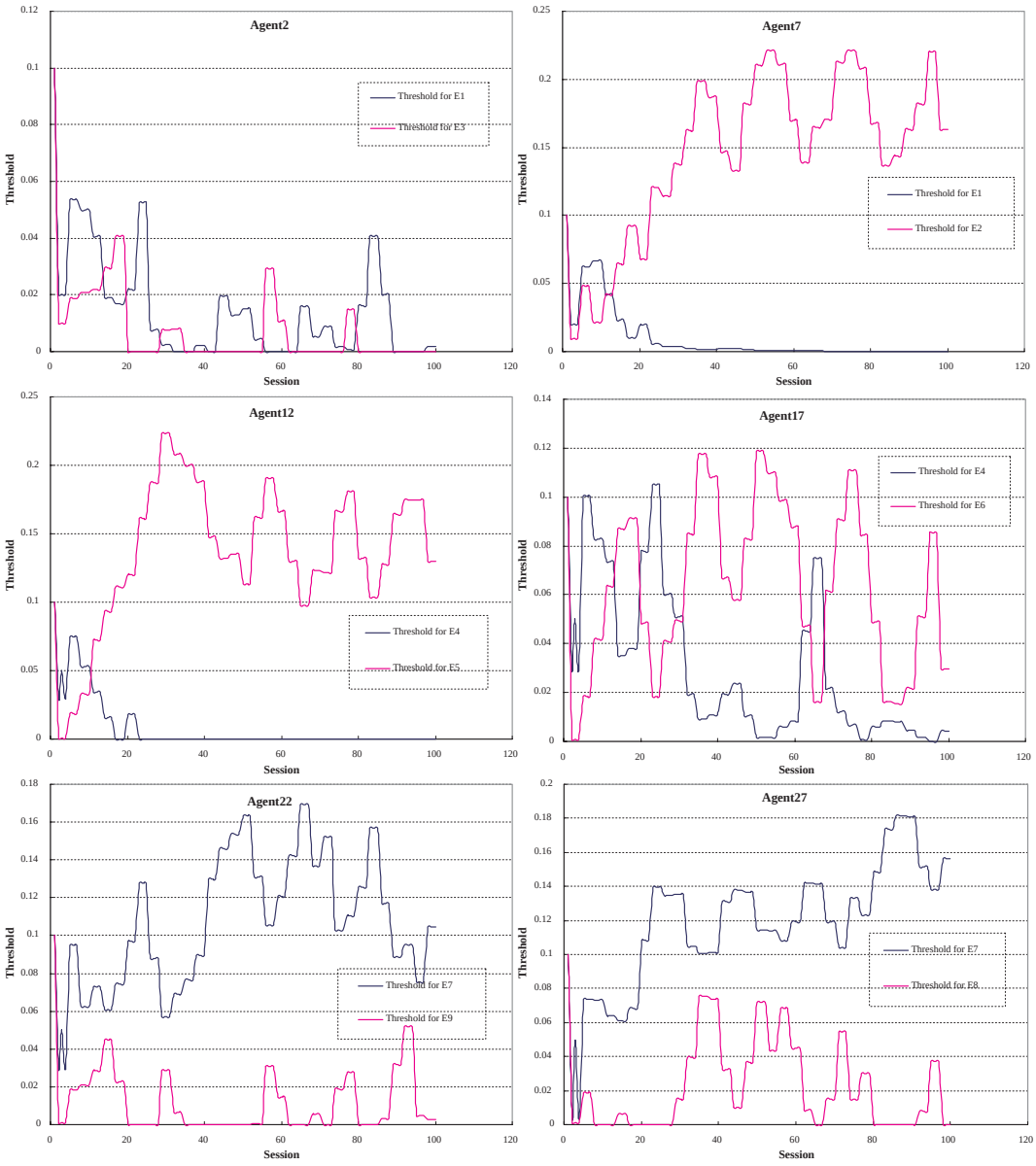

Figure 6.17: The changes of the local response thresholds of selected agents after using the metabolic coordination approach.

With all these settings, Figure 6.16 shows the corresponding system dynamics obtained. The changes of the response thresholds of selected agents are also depicted in Figure 6.17.

Figure 6.18 shows the amount of products manufactured and the system efficiency within each session from session 1 to 100. As indicated in Figure 6.18, within a major period of time, the system efficiency is above 0.9. As listed in Table 6.5, the average system efficiency between sessions 6 and 100 is 0.944. The average amounts of the 3 types of products manufactured are nearly identical, with a difference of only 0.38. Comparing with the experimental results given in Table 6.4, it is therefore evidenced that the metabolic coordination approach is ef-

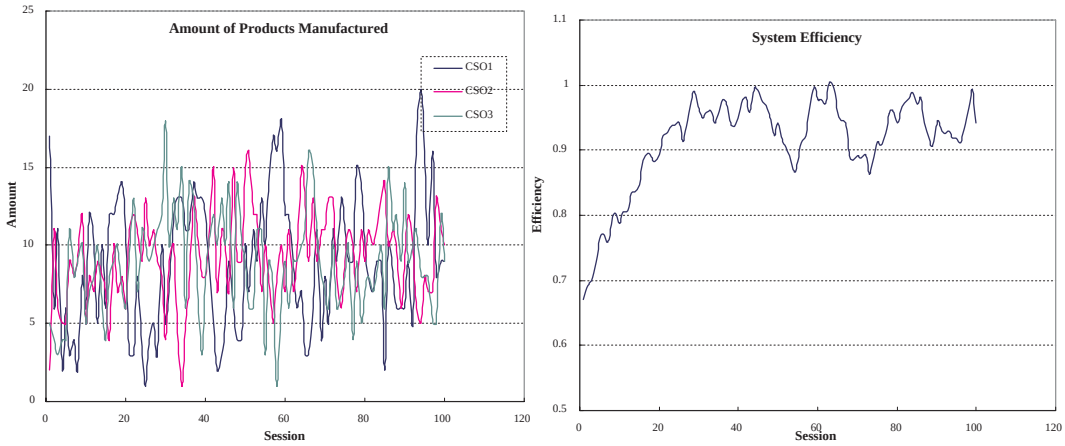

Figure 6.18: The amount of products manufactured and the system efficiency of each session when the coordination approach is applied.

fective in coordinating agents' behaviors as the inequality constraints are basically satisfied and the system efficiency is notably improved.

To further investigate the effectiveness of our coordination approach, comparison studies have been performed where agents use Genetic Algorithms [126] to optimize their θ changing policies. We regulate that all agents of the same group will take the same two policies for their two response thresholds. Each policy has a period 3. As the shop floor system includes 6 groups (Ref. Figure 6.12), a total of $2 \times 6 = 12$ policies are encoded by every chromosome. The Genetic Algorithm starts with an initial population of 30 chromosomes. For each chromosome, the 30 shop floor agents take the encoded policies and start operating from the same *combined density vector* $\vec{X}$. The amount of products manufactured between time steps 60 and 1000 is used to evaluate the *fitness* of the chromosome. Let $N(CSO_1)$, $N(CSO_2)$, and $N(CSO_3)$ denote the average amount of products produced during each session between time steps 60 and 1000. Let Avg be the *arithmetic mean* of $N(CSO_1)$, $N(CSO_2)$, and $N(CSO_3)$. The *fitness* is evaluated as

$$N(CSO_1) + N(CSO_2) + N(CSO_3) + 20 - 5 \cdot \sqrt{\sum_{k=1}^{3} \left(N(CSO_k) - Avg\right)^2}$$

It is easy to verify that the *fitness* reaches its maximum (50) when $N(CSO_1) = N(CSO_2) = N(CSO_3) = 10$. The higher the fitness, the more probable that the chromosome will be selected to produce its offspring. The Genetic Algorithm therefore works towards satisfying the three inequality constraints and improving the system efficiency as our previous experiment. The algorithm has been performed for 50 generations. The highest fitness encountered within each generation is depicted in Figure 6.19.

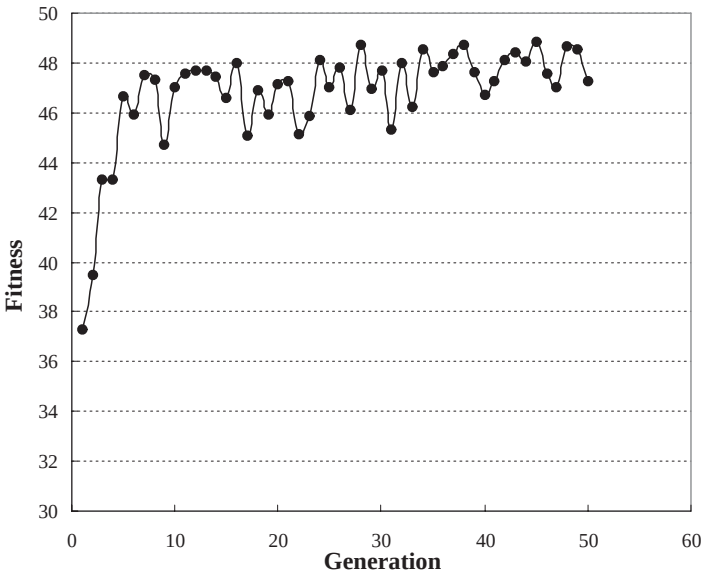

Figure 6.19: The highest fitness achieved for 50 consecutive generations.

The highest fitness achieved is 48.7, which appears at the 37-th generation of the Genetic Algorithm. Applying policies encoded in the corresponding chromosome, the amount of products produced and the system efficiency during each session between time steps 1 and 1000 are shown in Figure 6.20. Table 6.6 gives further the average amount of products manufactured and the average system efficiency between time steps 60 and 1000.

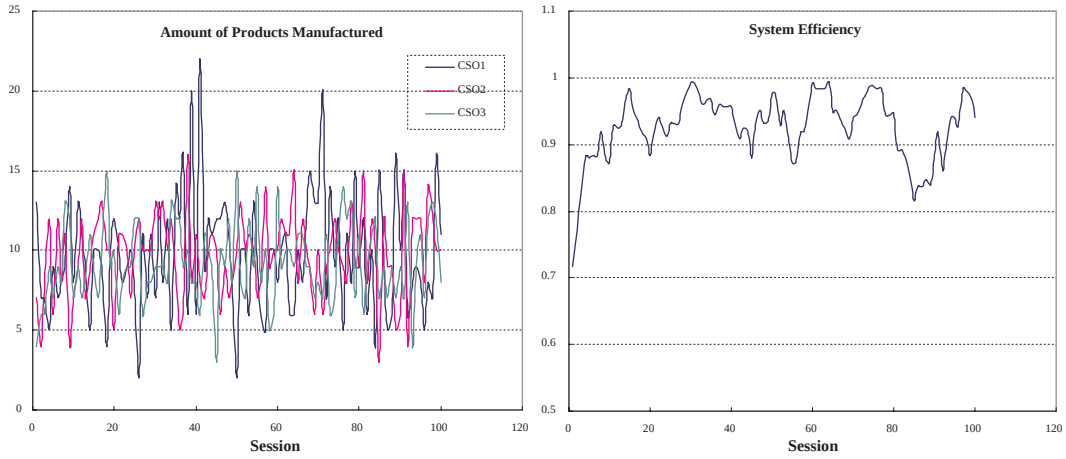

Figure 6.20: The amount of products manufactured and the system efficiency of each session after using policies identified by the Genetic Algorithm.

Table 6.6 indicates that policies identified through Genetic Algorithms may achieve a slightly higher system efficiency. But the difference among $N(CSO_1)$, $N(CSO_2)$, and $N(CSO_3)$ is enlarged from 0.38 to 0.53. It suggests that there exists a tradeoff between improving global performance and satisfying inequality

CSO_1	CSO_2	CSO_3	Efficiency
9.74	9.67	9.21	0.95

Table 6.6: The average amount of products manufactured and the average system efficiency after using policies identified by the Genetic Algorithm.

constraints. As our coordination approach focuses on the satisfaction of these constraints, it performs slightly better in this regard at the cost of achieving a little lower performance. Unlike Genetic Algorithms, which are used to identify policies in advance, our approach allows shop floor agents to adjust their local decisions based on system feedback. It is more flexible and computationally efficient. Agents' policies are constantly adjusted to respect the changing environment. And in each session, only a small amount of matrix operation is required.

6.6 Summary

In this Chapter, a metabolic approach for agent coordination was presented. It combines a dynamic coordination model with a systematic local adjustment process to be performed by each agent. Through this approach, the global system dynamics were explicitly utilized by agents in their decision making processes. It successfully helped agents achieve good global performance via their local interactions and behaviors.

The practicality of the proposed metabolic approach is investigated in a multi-agent shop floor system involving 30 agents, each of which manages a separate manufacturing device. The experiments demonstrated the basic procedure of applying the coordination approach in practical applications. The results indicated that the manufacturing process was well-coordinated and the proposed approach was effective. Our approach has no dependence on the correct function of any shop floor agents. For example, a stop failure of $Agent_{11}$ does not alter the structure of the metabolic network. This failure is considered as a change of global system dynamics, which will be recognized by agents in other groups to re-allocate their tasks adaptively. But to evaluate the exact impact of such failures, extensive experiments have to be performed and are left here as a future work.

Chapter 7

Conclusion and Recommendations

Coordination is the process of managing interdependencies between activities [190]. Without properly coordinating agents' behaviors, the full potential of the multi-agent system paradigm cannot be fulfilled. This thesis focused on developing agent coordination technologies in general task-oriented domains. The major research objective is to design and examine coordination mechanisms that enable a group of agents to accomplish their missions in a changing and interdependent environment. With this objective, four distinct contributions have been made and will be summarized in this chapter. This chapter also discusses promising directions of future research in this challenging research field.

7.1 Conclusion

This thesis explored agent coordination problems in task-oriented domains. It began with a brief introduction of the research motivations and objectives. After summarizing the various agent coordination techniques reported in the literature (Chapter 2), the major contributions were presented in consequent chapters (Chapter 3 to Chapter 6). A brief conclusion of each of these chapters are as follows.

- The Fuzzy Subjective Task Structure Model (FSTS) presented in Chapter 3 is used to describe agent's subjective beliefs and preferences in an uncertain environment. With this model, the information essential to agent coordination is effectively and explicitly modeled and incorporated by each agent to guide their local decision-making processes. This model is illustrated through a multi-agent pathology lab system.
- The two reinforcement learning algorithms, "coarse-grained" and "find-grained", are discussed in Chapter 4. They are proposed to solve the Decision Theo-

retic Planning (DTP) problem identified by the FSTS model. The "coarse-grained" algorithm operates at one level and tackles hard system constraints, while the "fine-grained" at another level and for soft constraints. These learning algorithms are formally proved to converge. Constraint information is explicitly incorporated into the reinforcement learning process. This approach towards system constraints underpins the two algorithms and attributes their effectiveness. The practicability of the two algorithms is investigated further using a multi-agent pathology lab system.

- In Chapter 5, a protocol for maintaining consistent system state information in a multi-agent environment is proposed to achieve effective reinforcement learning. The protocol properties were derived and theoretically analyzed. The experimental evaluations were conducted for a pursuit game in the context of two reinforcement learning algorithms. The results show that the protocol is effective and the reinforcement learning algorithms using it perform much better.

- Without using reinforcement learning, a bio-inspired approach to multi-agent coordination is described in Chapter 6. It combines a dynamic coordination model with a systematic local adjustment process to be performed by each agent. Chapter 6 showed that through this approach, the global system dynamics are explicitly utilized by every agent in their local decision-making processes. It successfully helped agents achieve good global performance via their local interactions and behaviors. The effectiveness of this approach is investigated in a multi-agent manufacturing environment involving 30 agents, each of which manages a separate manufacturing device. The experiments demonstrated the basic procedure of applying this coordination method in practical applications, The results evidenced that the manufacturing process was well-coordinated and the proposed approach was effective.

7.2 Recommendation for Future Work

In this section, future research directions regarding to contributions presented in Chapter 4 and 6 will be discussed. These contributions have been evaluated through selected application domains. An important future direction is their application to additional domains. Broad applications help verifying the generality of these methods as well as identifying their most important features for achieving successful results. We specifically list three application domains as examples.

- The problem of coordinating several robots (agents) that cooperatively ex-

plore certain terrain. The tasks of these robots are to collect and analyze samples using shared resources. To enhance cooperation, agents may divide the overall work among themselves. For example, one agent may focus on digging samples, while another agent will focus on the sample analysis. This work division is judged of a decision-making problem. When it is modeled using FSTS, agents can use the reinforcement learning algorithms proposed in the thesis to decide at each time what types of actions (e.g. digging or analyzing) they should pursue. On the other hand, when the problem is modeled as a metabolic system, agents can use the response threshold mechanism to determine a proper use of their resources according to the global system dynamics.

- The problem of network information sharing. Multiple agents residing in different network locations are needed to cooperatively share their local information in order to meet the end users' requirements. The tasks are to collect and process the exchanged information and present it to the end users. The information users need depends on the type of information and the way the information is organized. Therefore, each type of information collection and organization processing is considered as a separate action. When the problem is modeled using FSTS, agents can use reinforcement learning to decide which information processing action they should take in order to maximize each user's satisfaction on a statistical basis. By taking a metabolic approach, the coordination objective would be to enhance the information processing throughput and to improve the information processing efficiency.

- The problem of planning travels for a group of people. Providing a final travel plan for every people is considered as a separate task. In order to obtain such a plan, multiple travel planning agents are required to share their information and exploit the efforts of each other. For example, one agent might be responsible for checking the weather at the destination while another agent will in turn use this information to decide the specific places to visit. Again, the coordination problem can be modeled using FSTS. The coordination objectives and approaches are also similar to the discussions in the previous two examples.

7.2.1 Future Research for Agent Coordination via Reinforcement Learning

Using the FSTS model, the agent coordination can be boiled down to a decision problem and modeled as a Markov Decision Process (MDP). It led us to take a

DTP approach for agent coordination, where the reinforcement learning methods can be appropriately applied. But the learning process may be sensitive to the initial system conditions and the overall task structures. As a consequence, agents' decision policies may change wildly with slight changes of the task-oriented environments. Thus, another approximation function for estimating the changing process of agents' interdependencies may be necessary to be built into the FSTS model and then utilized explicitly by the learning algorithms. For example, if the relations (hard or soft) between the actions performed to achieve a task will change across time, agents might want to model and predict this change such that their decision policies could better adapt to the changing environments. Namely, the procedure that guides the adjustment of agents' local decision policies will become part of the overall learning objectives.

Although the two learning algorithms, the "coarse-grained" and the "fine-grained", has been demonstrated to be effective, in order to achieve an even higher global system performance, a structured approach that employs other decision-making mechanisms may be necessary. Several possibilities are listed as follows.

- In a complicated task-oriented domain where agents have to choose among a huge amount of potential actions at each decision point, it might be helpful to establish a hierarchical action structure over all possible actions. For example, at a certain decision point, an agent might first choose the type of actions (e.g. centrifugal or deposition operation in the pathology lab system). It then considers what devices to use (e.g. what centrifugal devices will be used in order to accomplish this action). After that, the agent will fix step by step other details for the purpose of performing an action (e.g. how long the centrifugal operation will take and whether certain chemical reagent should be added into the pathological sample). With this treatment, the complex problem of identifying a proper action with its full details is divided into several sub-problems, while each of these sub-problems is of limited complexity.

- One approach to effectively utilize the hierarchical action structure is to follow a layered learning process [256]. For example, both the "coarse-grained" and the "fine-grained" learning algorithms can be structured into three layers. At the highest layer, only the expected qualities of performing certain types of actions are to be approximated. At the middle layer, the qualities of using any particular devices will be estimated through a separate learning process. Finally, at the lowest layer, the duration of any given pathological operation will be further determined. For each learning problem in the three layers, the feature space includes only a small part of the features

used to describe an action, and the problem complexity is thus reduced. One issue regarding this layered learning approach is the *error propagation*. As each of the learned layers is achieved using a separate learning process, it is possible that errors at one layer could hurt performance at all other layers. But since all the learning processes are performed individually, it is also possible that the learning at one layer could compensate for the errors of other layers. A detailed study of error propagation and compensation serves as a promising area for future research.

- It is often a time-consuming process to identify a good approximation of the qualities of performing the various actions using a reinforcement learning method. In order to reduce the learning cost, it is better to combine the learning algorithms with other decision-making mechanisms. For example, a good decision policy can be determined *a priori* through genetic algorithms in a simulated environment. This decision policy will serve as the starting point for the learning algorithms in practical applications. Meanwhile, the learned results may be further incorporated into the simulated environment such that an even better decision policy can be in turn identified by the genetic algorithms. Another possibility is to use Bayesian learning algorithms to estimate the stochastic model of the task-oriented domain. This stochastic information can be utilized by the dynamic programming method to reach a decision policy that contributes to a high global performance.

- The learning algorithms can also be used to enhance the performance of other decision-making mechanisms. For instance, the decision to take any actions can be derived from planning or heuristic search processes. In this case, reinforcement learning may be utilized by the planning process to evaluate the performance of any given plans and in turn identify actions that contribute to the plan with the highest expected performance.

In the theoretical aspect, the *equivalence* notion may be formulated so that certain learning approaches reported in the literature can be shown actually *equivalent* to the methods proposed in this thesis. This subject has already been touched in Chapter 4. Specifically, two learning based coordination approaches are considered *equivalent* if the system models adopted or domain information utilized by the learning algorithms are identical. However, the notion was only vaguely described and illustrated with an example. To achieve some theoretical results, a thorough definition of the *equivalence* notion is necessary.

7.2.2 Future Research for Agent Coordination as Metabolic Systems

The work on agent coordination as dynamic metabolic systems (Chapter 6) opens doors for future research. At least two research issues require further investigation. They are described briefly as follows.

- **Changing global requirements**. In the bio-inspired approach towards multi-agent coordination, agent coordination is achieved when every agent performs iteratively a local adjustment process. The process follows the basic steps of typical dynamic optimization algorithms, but the coordination approach is designed for satisfying globally a group of inequality constraints and improving a pre-defined performance index. While numerical analysis demonstrates the effectiveness of the proposed approach, further study is required to see how effective the approach can be for continuously changing global requirements. For example, when the inequality constraints to be satisfied differ at separate moments, the coordination mechanism must adjust agents' local decision policies appropriately such that the global system dynamics respect these requirement changes. One way to achieve this is to identify and remember the policies used for different situations. Agents recognize and adopt different policies according to their policy repositories. But the procedure to transform from one policy to another still remains unsettled. Meanwhile, the impact of this memory based mechanism to the general stability of the proposed coordination approach requires extensive evaluation.

- **Changing metabolic networks**. In a task-oriented domain, such as a manufacturing system, the products to be manufactured might change regularly due to customers' changing demand. Such product change often alters the manufacturing process and therefore changes the metabolic network used by the proposed coordination approach. The challenge here would be how the coordination approach can adapt to such a changing metabolic network. More sophisticated coordination process is required for agents to recognize such network changes and adjust their local decision policies accordingly. To recognize network changes, every agent might maintain a separate copy of the metabolic network. When new products have been added into or old products removed from the manufacturing system, the agent who recognizes this change adjusts its metabolic network and propagates the modified network to other agents. To adapt to a changing metabolic network, memory based mechanisms, similar to the method used for handling changing global requirements, seem suitable. Nevertheless, the stability of such an approach

is unknown as well.

The situation becomes even more challenging when both of the two issues above arise at the same time. More efforts have to be made before this bio-inspired coordination approach can be applied to real-life applications with its full potential. From a theoretical perspective, although traditional dynamic optimization algorithms can be proved to converge, there lack theoretical guarantees when the environment is constantly changing. As a promising future research, preliminary theoretical results might be obtained to give more insights into this bio-inspired coordination method.

Publication List

Journal Paper

1. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Coordinating Multiple Agents via Reinforcement Learning," *Autonomous Agents and Multi-Agent Systems*, vol. 10, pp. 273–328, 2005.

2. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Maintaining system state information in a multiagent environment for effective learning," *IEICE Transactions on Information and Systems*, vol. E88-D, no. 1, pp. 127–134, 2005.

3. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Agent Coordination from a Dynamic Systems Perspective," *International Journal of Software Engineering and Knowledge Engineering (Under Review)*, 2005.

Conference Paper

1. Gang Chen, Zhonghua Yang, Simon See, and Jie Song, "Agent-mediated Genetic Super-scheduling in Grid Environments," in *Proceedings of the 5th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2004)*, 2004.

2. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Coordinating Multiple Agents via Reinforcement Learning,", in *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003)*, 2003.

3. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Multiagent Coordination: A Fuzzy Logic Based Approach," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2003)*, 2003.

4. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "A Fuzzy-Logic Based Multi-Agent Coordination Framework," in *Proceedings of the Inter-*

national Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2003), 2003.

5. Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh, "Coordinating Multi-Agents using JavaSpaces," in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS 2002)*, 2002.

Bibliography

[1] "Specifications, Foundations for Intelligent Physical Agents," http://www.fipa.org, 2000.

[2] "Fipa acl message structure specification," http://www.fipa.org/specs/fipa00061/, 2002.

[3] O. Abul, F. Polat, and R. Alhajj, "Multiagent reinforcement learning using function approximation," in *IEEE Transactions on Systems, Man, and Cybernetics*, 2000, pp. 485–497.

[4] M. R. Adler, A. B. Davis, R. Weihmayer, and R. W. Worrest, "Conflict-resolution strategies for nonhierarchical distributed agents," in *Distributed Artificial Intelligence*, vol. 2. Morgan Kaufmann, 1989, pp. 139–161.

[5] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and friends," *IEEE Computer*, vol. 19, no. 8, pp. 26–34, 1986.

[6] M. R. Alder, A. B. Davis, R. Weihmayer, and R. W. Forest, "Conflict-resolution strategies for non-hierarchical distributed agents," in *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, Eds., vol. II. Morgan Kaufmann, 1989.

[7] S. Arai, K. Sycara, and T. R. Payne, "Multi-agent Reinforcement Learning for Scheduling Multiple-Goals," in *Proceedings of the 4th International Conference on Multi-Agent Systems*, 2000.

[8] F. Arbab, "The IWIM Model for Coordination of Concurrent Activities," in *First International Conference on Coordination Models, Languages and Applications (Coordination' 96)*. Springer-Verlag, 1996, pp. 34–56.

[9] F. Arbab, I. Herman, and P. Spilling, "An overview of Manifold and its implementation," *Concurrency: Practice and Experience*, vol. 5, no. 1, pp. 23–70, 1993.

[10] J. L. Austin, *How To Do Things With Words*. Oxford University Press, 1962.

[11] R. Axelrod, *The Evolution of Cooperation.* Basic Books, 1985.

[12] P. Bak, *How Nature Works: The Science of Self-Organized Criticality.* Copernicus Books, 1996.

[13] A. D. Baker, "Manufacturing control with a market-driven contract net," Ph.D. dissertation, Electrical, Computer and Systems Engineering Department, Troy, NY: Rensselaer Polytechnic Institute, 1991.

[14] T. Balch and R. C. Arkin, "Motor schema-based formation control for multiagent robot teams," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, pp. 10–16.

[15] J. P. Banatre and D. LeMetayer, "The gamma model and its discipline of programming," *Science of Computer Programming*, vol. 15, pp. 55–77, 1990.

[16] M. Barbuceanu and M. S. Fox, "Cool: A language for describing coordination in multiagent systems," in *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[17] R. A. Barman, S. J. Kingdon, J. J. Little, A. K. Mackworth, D. K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang, "DYNAMO: Real-time experiments with multiple mobile robots," in *Intelligent Vehicles Symposium*, 1993, pp. 261–266.

[18] R. Becker, S. Zilberstein, and V. Lesser, "Decentralized Markov Decision Processes with Event-Driven Interactions," in *The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.

[19] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman, "Transition-independent decentralized markov decision problems," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003.

[20] ——, "Solving transition independent decentralized markov decision processes," *Journal of Artificial Intelligence Research*, pp. 423–455, 2004.

[21] R. D. Beer, "A dynamical systems perspective on agent-environment interaction," *Artificial Intelligence*, pp. 173–215, 1995.

[22] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE – a FIPA-compliant agent framework," in *Proc. Practical Applications of Intelligent Agents and Multi-Agents*, 1999, pp. 97–108.

[23] R. Bellman, *Dynamic Programming.* Princeton University Press, 1957.

[24] M. Benda, R. Jagganathan, and R. Dodhiawalla, "On optimal cooperation of knowledge sources," in *Unpublished manuscript presented at the 1986 Workshop on Distributed Artificial Intelligence*, 1986.

[25] H. R. Berenji and D. Vengerov, "Cooperation and Coordination Between Fuzzy Reinforcement Learning Agents in Continuous State Partially Observable Markov Decision Processes," in *Proceedings of the 8th IEEE International Conference on Fuzzy Systems*, 1999, pp. 621–627.

[26] F. Bergenti and A. Ricci, "Three approaches in coordination of MAS," in *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'02)*, 2002.

[27] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of Operations Research*, pp. 819–840, 2002.

[28] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[29] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[30] V. A. Bharatia and D. J. Cook, "Design and analysis of centralized, distributed, and group multi-agents coordination models," Department of Computer Science and Engineering, University of Texas at Arlington, Tech. Rep., 1995.

[31] R. Bjornson, "Linda on distributed memory multiprocessors," Ph.D. dissertation, Yale University, 1992.

[32] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[33] E. Bonabeau, G. Theraulaz, and J. L. Deneubourg, "Fixed response thresholds and the regulation of division of labor in insect societies," *Bulletin of Mathematical Biology*, pp. 753–807, 1998.

[34] A. Bonarini and V. Trianni, "Learning fuzzy classifier systems for multi-agent coordination," *International Journal of Information Sciences*, pp. 215–239, 2001.

[35] A. H. Bond and L. Gasser, "Readings in distributed artificial intelligence," *Morgan Kaufmann*, 1988.

[36] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[37] C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artificial Intelligence*, vol. 121, pp. 49–107, 2000.

[38] M. Bowling, B. Browning, and M. Veloso, "Plays as effective multiagent plans enabling opponent-adaptive play selection," in *Proceedings of International Conference on Automated Planning and Scheduling*, 2004.

[39] M. Bowling and M. Veloso, "Simultaneous adversarial multi-robot learning," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[40] P. Bretier and D. Sadek, "A rational agent as the kernel of a cooperative spoken dialogue system: implementing a logical theory of interaction," in *Intelligent Agents, III*, J. P. Müller, M. Wooldridge, and N. R. Jennings, Eds. Springer, 1997, pp. 189–204.

[41] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, pp. 14–23, 1986.

[42] A. E. Bryson, *Dynamic Optimization*. Addison-Wesley, 1999.

[43] H. H. Bui, S. Venkatesh, and D. Kieronska, "Learning other agents' preferences in multi-agent negotiation using the bayesian classifier," in *International Journal of Cooperative Information Systems*. World Scientific Publishing Company, 1999, pp. 275–293.

[44] H. Bunke and X. Jiang, "Graph matching and similarity," in *Intelligent Systems and Interfaces*, H. N. Teodorescu, D. Mlynek, A. Kandel, and H. J. Zimmermann, Eds. Kluwer Academic Publishers, 2000.

[45] P. Busetta, M. Merzi, S. Rossi, and F. Legras, "Intra-role coordination using group communication: a preliminary report," in *Workshop on Agent Communication Languages*, 2003, pp. 231–253.

[46] N. Busi, P. Ciancarini, R. Gorrieri, and G. Zavattaro, *Coordination of Internet Agents*. Springer-Verlag, 2001, ch. Coordination Models: A Guided Tour.

[47] S. Bussmann, "Agent-oriented programming of manufacturing control tasks," in *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS-1998)*, 1998, pp. 57–63.

[48] S. Bussmann and H. J. Muller, "A negotiation framework for cooperating agents," in *Proc. Spec. Interest Group Coop. Knowl. Based Syst., SIG Workshop*, 1992, pp. 1–18.

[49] S. Bussmann and K. Schild, "Self-Organizaing Manufacturing Control: An Industrial Application of Agent Technology," in *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, 2000, pp. 87–94.

[50] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of cooperation in distributed problem solving," in *Proc. of the Eighth Int. Joint Conf. on Artificial Intelligence*, 1983, pp. 767–770.

[51] M. Campos, E. Bonabeau, and G. Theraulaz, "Dynamic scheduling and division of labor in social insects," *Adaptive Behavior*, vol. 8, no. 2, pp. 83–96, 2000.

[52] J. Carreria, L. Silva, and J. Silva, "On the design of eilean: A linda-like library for mpi," Universidade de Coimbra, Tech. Rep., 1994.

[53] N. Carriero, "Implementation of tuple space machines," Ph.D. dissertation, Yale University, 1987.

[54] N. Carriero and D. Gelernter, "How to write parallel programs: A guide to the perplexed," *ACM Computing Surveys*, vol. 21, no. 3, pp. 323–357, 1989.

[55] ——, "Tuple analysis and partial evaluation strategies in the linda precompiler," in *Languages and Compilers for Parallel Computing*. MIT Press, 1990, pp. 114–125.

[56] G. Chalkiadakis and C. Boutilier, "Coordination in Multiagent Reinforcement Learning: A Bayesian Approach," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, 2003, pp. 709–716.

[57] G. Chen, Z. Yang, H. He, and K. M. Goh, "Coordinating Multi-Agents using JavaSpaces," in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS 2002)*, 2002.

[58] ——, "A Fuzzy-Logic Based Multi-Agent Coordination Framework," in *Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2003)*, 2003.

[59] ——, "Coordinating Multiple Agents via Reinforcement Learning," in *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003)*, 2003.

[60] ——, "Multiagent Coordination: A Fuzzy Logic Based Approach," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, 2003.

[61] ——, "Agent coordination from a dynamic systems perspective," *International Journal of Software Engineering and Knowledge Engineering (Under Review)*, 2005.

[62] ——, "Coordinating multiple agents via reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 10, pp. 273–328, 2005.

[63] ——, "Maintaining system state information in a multiagent environment for effective learning," *IEICE Transactions on Information and Systems*, 2005.

[64] G. Chen, Z. Yang, S. See, and J. Song, "Agent-mediated Genetic Superscheduling in Grid Environments," in *Proceedings of the 5th International Conference on Parallel and Distributed Computing, Applicatons and Technologies (PDCAT 2004)*, 2004.

[65] H. Chen, *Stochastic approximation and its applications*. Kluwer Academic Publishers, 2002.

[66] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next generation high-speed networks: Problems and solutions," *IEEE Network*, vol. 12, pp. 64–79, 1998.

[67] A. Chimaris and G. A. Papadopoulos, "Control-driven coordination based assembling of components," in *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02)*, 2002.

[68] H. W. Chun and R. Y. M. Wong, "N* – an agent-based negotiation algorithm for dynamic scheduling and rescheduling," *Advanced Engineering Informatics*, pp. 1–22, 2003.

[69] P. Ciancarini and D. Rossi, "Coordinating java agents over the www," *World Wide Web Journal*, vol. 1, no. 2, pp. 87–99, 1998.

[70] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche, "Coordinating multiagent applications on the www: A reference architecture," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 362–375, 1998.

[71] P. Ciancarini, "Coordination models and languages as software integrators," *ACM Computing Surveys*, 1996.

[72] V. Cicirello and S. Smith, "Insect Societies and Manufacturing," in *The IJCAI-01 Workshop on Aritificial Intelligence and Manufacturing: New AI Paradigms for Manufacturing*, 2001.

[73] V. A. Cicirello and S. F. Smith, "Ant colony control for autonomous decentralized shop floor routing," in *ISADS-2001: International Symposium Autonomous Decentralized Systems*, 2001, pp. 383–390.

[74] ——, "Amplification of search performance through randomization of heuristics," in *The 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, 2002.

[75] ——, "Wasp-like agents for distributed factory coordination," *Autonomous Agents and Multi-Agent Systems*, vol. 8, pp. 237–266, 2004.

[76] P. Clayton and E. Wentworth, "Placing processes in a transputer-based linda programming environment," Rhodes University, Tech. Rep., 1992.

[77] P. R. Cohen and H. J. Levesque, "Rational interaction as the basis for communication," in *Intentions in Communication*, P. R. Cohen, J. Morgan, and M. E. Pollack, Eds. MIT Press, 1990, pp. 221–256.

[78] P. R. Cohen and C. R. Perrault, "Elements of a plan based theory of speech acts," *Cognitive Science*, vol. 3, pp. 177–212, 1979.

[79] J. Collins, M. Tsvetovatyy, B. Mobasher, and M. Gini, "MAGNET: A multiagent contracting system for plan execution," in *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, 1998, pp. 63–68.

[80] S. E. Conry, K. Kuwabara, and V. R. Lesser, "Multistage negotiation for distributed constraint satisfaction," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, 1991, pp. 1462–1477.

[81] R. S. Cost, Y. Labrou, and T. Finin, "Coordinating agents using agent communication languages conversations," in *Coordination of Internet Agents*. Springer-Verlag, 2001.

[82] E. Crawford and M. Veloso, "Learning to select negotiation strategies in multi-agent meeting scheduling," in *Proceedings of the 12th Portugese Conference on Artificial Intelligence*, 2005.

[83] R. H. Crites, "Large-scale dynamic optimization using teams of reinforcement learning agents," Ph.D. dissertation, University of Massachusetts Amherst, 1996.

[84] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine Learning*, vol. 33, no. 2, pp. 235–262, 1998.

[85] M. W. D. Deugo and E. Kendall, *Coordination of Internet Agents*. Springer, 2001, ch. Reusable Patterns for Agent Coordination.

[86] R. Dawkins, "Hierarchical organisation: A candidate principle for ethology," in *Growing Points in Ethology*, P. Bateson and R. Hinde, Eds. Cambridge University Press, 1976.

[87] M. M. de Weerdt, A. Bos, J. Tonino, and C. Witteveen, "A resource logic for multi-agent plan merging," *Annals of Mathematics and Artificial Intelligence, Special Issue on Computational Logic in Multi-Agent Systems*, pp. 93–130, 2003.

[88] R. Dearden and C. Boutilier, "Abstraction and approximate decision-theoretic planning," *Artificial Intelligence*, vol. 89, pp. 219–283, 1997.

[89] K. S. Decker, "Distributed problem solving: A survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 5, pp. 729–740, 1987.

[90] ——, "Environment centered analysis and design of coordination mechanisms," Ph.D. dissertation, University of Massachusetts Amherst, 1995.

[91] K. S. Decker and V. R. Lesser, "Generalizing the partial global planning algorithm," in *International Journal of Intelligent Cooperative Information Systems*, 1992, pp. 319–346.

[92] K. S. Decker and J. Li, "Coordinating mutually exclusive resources using gpgp," *Autonomous Agents and Multi-Agent Systems*, pp. 113–157, 2000.

[93] E. Denti and A. Omicini, "An architecture for tuple-based coordination of multi-agent systems," *Software – Practice & Experience*, vol. 29, no. 12, pp. 1103–1121, 1999.

[94] R. L. Devaney, *An introduction to chaotic dynamical systems*. Addison-Wesley, 1989.

[95] W. E. Dixon, S. Nagarkatti, D. Dawson, and A. Behal, *Nonlinear Control of Engineering Systems: A Lyapunov-Based Approach.* Birkhauser Boston, 2003.

[96] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[97] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[98] M. Dorigo and T. Stützle, *Ant Colony Optimization.* The MIT Press, 2004.

[99] A. Douglas, A. Wood, and A. Rowstron, *Transputer and occam developments.* IOS Press, 1995, ch. Linda implementation revisited, pp. 125–138.

[100] E. H. Durfee, *Coordination of distributed problem solvers.* Kluwer Academic Publishers, 1988.

[101] ——, "Distributed Problem Solving and Planning," in *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. The MIT Press, 1999, pp. 121–164.

[102] ——, "Scaling up agent coordination strategies," *IEEE Computer*, vol. 34, no. 7, pp. 39–46, 2001.

[103] E. H. Durfee and V. R. Lesser, "Negotiation task decomposition and allocation using partial global planning," in *Distributed Artificial Intelligence*, vol. 2. Morgan Kaufmann, 1989, pp. 229–243.

[104] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Transactions on Computers*, vol. 36, no. 11, pp. 1275–1291, 1987.

[105] ——, "Trends in cooperative distributed problem solving," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 63–83, 1989.

[106] E. H. Durfee and T. A. Montgomery, "Coordination as distributed search in a hierarchical behavior space," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1363–1378, 1991.

[107] E. H. Durfee, "Organizations, plans, and schedules: An interdisciplinary perspective on coordinating ai agents," *Journal of Intelligent Systems*, 1991.

[108] A. Dutech, "Solving pomdps using selected past events," in *Proc. of the 14th European Conference on Artificial Intelligence, ECAI2000*, 2000, pp. 281–285.

[109] E. Ephrati and J. S. Rosenschein, "Divide and conquer in multi-agent planning," in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, 1994, pp. 375–380.

[110] P. Faratin, C. Sierra, and N. R. Jennings, "Using similarity criteria to make negotiation trade-offs," in *Proceedings of the Fourth International Conference on Multi-Agent Systems*, 2000, pp. 119–126.

[111] M. Feng, Y. Gao, and C. Yuen, "Implementing linda tuple space on a distributed system," *International Journal of High Speed Computing*, vol. 7, no. 1, pp. 125–144, 1995.

[112] F. Fernandez and L. E. Parker, "Learning in large co-operative multi-robot domains," *International Journal of Robotocis and Automation*, pp. 217–226, 2001.

[113] R. E. Fikes and N. Nilsson, "Strips: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[114] T. Finin, "Specification of the KQML agent communication language," DARPA knowledge sharing initiative external interfaces working group, 1993.

[115] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, 1999.

[116] L. Gasser, C. Braganza, and N. Hermann, "Implementing distributed AI systems using MACE," in *Proceedings of the 3rd IEEE Conference on AI Applications*, 1987, pp. 315–320.

[117] L. Gasser, N. F. Rouquette, R. W. Hill, and J. Lieb, "Representing and Using Organizational Knowledge in Distributed AI Systems," in *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, Eds., vol. II. Morgan Kaufmann, 1989, pp. 55–78.

[118] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[119] D. Gelernter and N. Carriero, "Coordination languages and their significance," in *Communications ACM*, 1992.

[120] M. R. Genesereth and R. E. Fikes, "Knowledge Interchange Format, Version 3.0 Reference Manual," Computer Science Department, Stanford University, Tech. Rep., 1992.

[121] M. R. Genesereth, M. Ginsberg, and J. S. Rosenschein, "Cooperation without communication," in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, 1986, pp. 51–57.

[122] M. Georgeff, "A Theory of Action for Multi-Agent Planning," in *Proceedings of the National Conference on Artificial Intelligence*, 1984.

[123] M. Ghavamzadeh and S. Mahadevan, "A multiagent reinforcement learning algorithm by dynamically merging Markov decision processes," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002.

[124] E. Gimenez-Funes, L. Godo, and e. a. J. Rodriguez-Aguilar, "Designing bidding strategies for trading agents in electronic auctions," in *Proceedings of the Third International Conference on Multi-Agent Systems*, 1998, pp. 136–143.

[125] R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in markov decision processes," *Artificial Intelligence*, vol. 147, pp. 163–223, 2003.

[126] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[127] C. Goldman and J. Rosenschein, "Emergent coordination through the use of cooperative state-changing rules," in *Proceedings of the 12th National Conference on Artificial Intelligence*. Morgan Kaufman, 1994, pp. 408–413.

[128] C. V. Goldman and S. Zilberstein, "Optimizing information exchange in cooperative multi-agent systems," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.

[129] S. Y. Goldsmith and L. D. Interrante, "An autonomous manufacturing collective for job shop scheduling," in *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, 1998, pp. 69–74.

[130] C. Guestrin, S. Venkataraman, and D. Koller, "Context Specific Multiagent Coordination and Planning with Factored MDPs," in *AAAI Spring Symposium on Collaborative Learning Agents*, Stanford, California, March 2002.

[131] V. Hatzimanikatis, "Nonlinear metabolic control analysis," *Metabolic Engineering*, pp. 75–87, 1999.

[132] S. C. Hayden, C. Carrick, and Q. Yang, "A catalog of agent coordination patterns," in *Proceedings of the third annual conference on Autonomous Agents*, 1999, pp. 412–413.

[133] M. He, H. F. Leung, and N. R. Jennings, "A fuzzy logic based bidding strategy for autonomous agents in continuous double auctions," in *To Appear in IEEE Trans. on Knowledge and Data Engineering*, 2003.

[134] M. Heger, "Consideration of risk in reinforcement learning," in *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 105–111.

[135] F. Heylighen, "Evolution, selfishness and cooperation," *Journal of Ideas*, vol. 2, no. 4, pp. 70–76, 1992.

[136] F. Ho and M. Kamel, "Learning coordination strategies for cooperative multiagent systems," in *Machine Learning*. Kluwer Academic Publishers, 1998, pp. 155–177.

[137] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of AI Research*, pp. 253–302, 2001.

[138] C. Hoile, F. Wang, E. Bonsma, and P. Marrow, "Core Specification and Experiments in DIET: a Decentralised Ecosystem-inspired Mobile Agent System," in *Proceedings of the first International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, 2002, pp. 623–630.

[139] O. E. Holland, "Multiagent systems: Lessons from social insects and collective robotics," in *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*. AAAI Press, 1996, pp. 57–62.

[140] J. Honerkamp, *Stochastic dynamical systems: concepts, numerical methods, data analysis*. VCH Publishers, 1994.

[141] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, 1985.

[142] J. Hu, "Learning in dynamic noncooperative multiagent systems," Ph.D. dissertation, University of Michigan, 1999.

[143] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proc. of the 15th International Conference on Machine Learning*, 1998, pp. 242–250.

[144] M. Huhns and M. P. Singh, "Ckbs-94 tutorial: Distributed artificial intelligence for information systems," Dake Centre, University of Keele, England, 1994.

[145] M. Humphrys, "W-learning: Competition among selfish q-learners," University of Cambridge, Tech. Rep., 1995.

[146] N. Jennings, *Cooperation in Industrial Multi-Agent Systems*. World Scientific, 1994.

[147] N. R. Jennings, "Commitments and conventions: the foundation of coordination in multi-agent systems," *The Knowledge Engineering Review*, vol. 8, no. 3, pp. 223–250, 1993.

[148] ——, *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, 1996, ch. Coordination Techniques for Distributed Artificial Intelligence.

[149] ——, "On agent-based software engineering," *Artificial Intelligence*, vol. 117(2), pp. 277–296, 2000.

[150] N. R. Jennings, J. M. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriollat, P. Skarek, and L. Z. Varga, "Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control," *IEEE Expert*, vol. 11, no. 6, pp. 60–88, 1996.

[151] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated negotiation: Prospects, methods and challenges," in *Int. Journal of Group Decision and Negotiation*, 2000.

[152] K. Jeong, "Fault-tolerant parallel processing combining linda, checkpointing and transactions," Ph.D. dissertation, New York University, 1996.

[153] Y. Jin and T. Koyama, "Multi-agent planning through expectation based negotiation," in *Proceedings of the 10th International Workshop on DAI*, 1990.

[154] R. E. Kalman, "The Calculus of Variations and Optimal Control Theory," in *Mathematical Optimization Techniques*, R. Bellman, Ed. University of California Press, Berkeley, 1963.

[155] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Eighteenth national conference on Artificial intelligence*, Edmonton, Alberta, Canada, 2002, pp. 326–331.

[156] K*ö*n*ö*nen, "Multiagent reinforcement learning in markov games: Asymmetric and symmetric approaches," Ph.D. dissertation, Department of Computer Science and Engineering, Helsinki University of Technology, 2004.

[157] H. J. Kelley, "Gradient theory of optimal flight paths," *American Rocket Society Journal*, vol. 30, pp. 947–954, 1960.

[158] K. Kempf and T. Beaumariage, "Chaotic behavior in manufacturing systems," in *AAAI-94 Workshop Program, Reasoning About the Shop Floor, Workshop Notes*, 1994, pp. 82–96.

[159] E. A. Kendall, P. V. M. Krishna, C. B. Suresh, and C. V. Pathak, "An application framework for intelligent and mobile agents," in *ACM Computing Surveys*, vol. 32, no. 1es, 2000.

[160] J. O. Kephart, T. Hogg, and B. A. Huberman, "Dynamics of Computational Ecosystems: Implications for DAI," in *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, Eds., vol. II. Morgan Kaufmann, 1989, pp. 79–95.

[161] J. E. Kittock, "Emergent conventions and the structure of multi-agent systems," in *Proceedings of the 1993 Santa Fe Institute Complex Systems Summer School*, 1993.

[162] M. Klusch and K. Sycara, *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, 2001, ch. Brokering and Matchmaking for Coordination of Agent Societies: A Survey, pp. 197–224.

[163] R. Kowalczyk, "Fuzzy e-negotiation agents," in *Soft Computing 6*. Springer-Verlag, 2002, pp. 337–347.

[164] S. Kraus, "The strategic-negotiation model," in *Strategic Negotiation in Multiagent Environments*. MIT Press, 2001, pp. 17–27.

[165] S. Kraus and V. Subrahmanian, "Multiagent reasoning with probability, time and beliefs," in *International Journal of Intelligent Systems*, vol. 10, no. 5, 1995, pp. 459–499.

[166] T. Kreifelt and F. von Martial, "A negotiation framework for autonomous agents," in *Decentralized AI2*, Y. Demazeau and J. P. Muller, Eds. Elsevier Science, 1991.

[167] M. R. S. Kulenović and O. Merino, *Discrete Dynamical Systems and Difference Equations with Mathematica*. Chapman & Hall/CRC, 2002.

[168] Y. Labrou, "Semantics for an agent communication language," Ph.D. dissertation, University of Maryland Baltimore County, 1996.

[169] Y. Labrou and T. Finin, "Semantics and conversations for an agent communication language," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[170] Y. Labrou, T. Finin, and Y. Peng, "The current landscape of agent communication languages," *Intelligent Systems, IEEE Computer Society*, vol. 14, no. 2, 1999.

[171] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[172] S. Lander and V. R. Lesser, "Understanding the role of negotiation in distributed search among heterogeneous agents," in *Proc. Int. Workshop Distributed Artif. Intell., 12th, Hidden Valley, PA*, 1993, pp. 249–262.

[173] E. Laszlo, *Introduction to Systems Philosophy*. Gordon and Breach, Science Publishers, 1972.

[174] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. of the 17th International Conference on Machine Learning*, 2000, pp. 535–542.

[175] J. Leichter, "Shared tuple memories, shared memories, buses and lan's – linda implementations across the spectrum of connectivity," Ph.D. dissertation, Yale University, 1989.

[176] V. Lesser and D. Corkill, "The distributed vehicle monitoring testbed: A tool for investigating distributed problem-solving networks," *AI Magazine*, vol. 4, no. 3, pp. 15–33, 1983.

[177] ——, "Distributed problem solving," in *Encyclopedia of Artificial Intelligence*, S. Shapiro and D. Eckroth, Eds., 1987.

[178] V. R. Lesser, "Multiagent systems: An emerging subdiscipline of ai," *ACM Computing Surveys*, vol. 27, no. 3, pp. 340–342, 1995.

[179] V. R. Lesser and D. D. Corkill, "Functionally accurate, cooperative distributed systems," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1981, pp. 81–96.

[180] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes, "On acting together," in *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, 1990, pp. 94–99.

[181] D. Lewis, *Convention - a Philosophical Study*. Harvard University Press, 1969.

[182] S. Li, *Professional Jini*. Wrox Press, 2000.

[183] G. Y. J. Lin and J. J. Solberg, "Integrated shop floor control using autonomous agents," *IIE Transactions*, vol. 24, no. 3, pp. 57–71, 1992.

[184] J. S. Liu, "Coordination of multiple agents in distributed manufacturing scheduling," Ph.D. dissertation, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[185] L. Ljung, *System Identification: Theory for the User*, 2nd ed. Prentice Hall, 1998.

[186] X. Luo, N. R. Jennings, N. Shadbolt, H. F. Leung, and J. H. M. Lee, "A fuzzy constraint based knowledge model for bilateral, multi-issue negotiations in competitive environments," in *Working Paper, Dept. of Electronics and Computer Science, the University of Southampton*, 2001.

[187] P. Maes and R. A. Brooks, "Learning to Coordinate Behaviors," in *Proceedings of the National Conference on Artificial Intelligence*, 1990, pp. 796–802.

[188] G. Maione and D. Naso, "A genetic approach for adaptive multiagent control in heterarchical manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 33, no. 5, 2003.

[189] T. W. Malone, "Modeling coordination in organizations and markets," *Management Science*, vol. 33, no. 10, pp. 1317–1332, 1987.

[190] T. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Computing Surveys*, 1994.

[191] M. J. Mataric, "Issues and approaches in the design of collective autonomous agents," *Robotics and Autonomous Systems*, vol. 16, pp. 321–331, 1995.

[192] ——, "Learning in behavior-based multi-robot systems: policies, models, and other agents," *Journal of Cognitive Systems Research*, pp. 81–93, 2001.

[193] T. Matsuo and T. Ito, "A Group Formation Support System based on Substitute Goods in Group Buying," in *Systems and Computers in Japan*, vol. 35, no. 10. John Wiley & Sons, 2004, pp. 23–31.

[194] D. Morley, "Painting trucks at general motors: The effectiveness of a complexity-based approach," in *Embracing complexity: Exploring the Application of Complex Adaptive Systems to Business*. The Ernst and Young Center for Business Innovation, 1996, pp. 53–58.

[195] D. Morley and C. Schelberg, "An analysis of a plant-specific dynamic scheduler," in *Proceedings of the NSF Workshop on Dynamic Scheduling*, 1993, pp. 115–122.

[196] T. E. Morton and D. W. Pentico, *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley and Sons, 1993.

[197] H. J. Muller, "Negotiation principles," in *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, INC., 1996, pp. 211–230.

[198] J. R. Munkres, *Elements of Algebraic Topology*. Perseus Press, 1993.

[199] R. D. Nicola, G. L. Ferrari, and R. Pugliese, "Klaim: A kernel language for agents interaction and mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 315–330, 1998.

[200] O. Nierstrasz and D. Tsichritzis, Eds., *Object-Oriented Software Composition*. Prentice-Hall, 1995.

[201] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[202] H. S. Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis, "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems," in *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, 1999.

[203] H. Nwana, L. Lee, and N. Jennings, "Coordination in software agent systems," *The British Telecom Technical Journal*, 1996.

[204] T. Oates, M. V. N. Prasad, and V. R. Lesser, "Cooperative information gathering: A distributed problem-solving approach," in *IEEE Proceedings on Software Engineering, Special Issue on Agent-based Systems*, vol. 144, no. 1, January 1997, pp. 72–88.

[205] A. Omicini and S. Ossowski, "Objective versus subjective coordination in the engineering of agent systems," in *Intelligent Information Agents: An AgentLink Perspective*, M. Klusch, S. Bergamaschi, P. Edwards, and P. Petta, Eds. Springer-Verlag, 2003, pp. 179–202.

[206] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination Artifacts: Environment-based Coordination for Intelligent Agents," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 1, 2004, pp. 286–293.

[207] A. Omicini, A. Ricci, M. Viroli, and G. Rimassa, "Integrating Objective & Subjective Coordination in Multi-Agent Systems," in *ACM Symposium on Applied Computing*, 2004, pp. 449–455.

[208] A. Omicini and F. Zambonelli, "Coordination for internet application development," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, 1999.

[209] S. Ontanon and E. Plaza, "Learning when to collaborate among learning agents," in *ECML 2001*, D. Raedt and P. Flach, Eds. Springer-Verlag, 2001, pp. 394–405.

[210] M. J. Osborne, *An Introduction to Game Theory*. Oxford University Press, 2003.

[211] S. Ossowski, *Social Structure and Its Implications for Autonomous Problem-Solving Agents*. Springer-Verlag, 1999, ch. Co-ordination in Artificial Agent Societies.

[212] G. Papadopoulos and F. Arbab, "Coordination models and languages," *Advances in Computers*, vol. 46, 1998.

[213] G. A. Papadopoulos, *Coordination of Internet Agents*. Springer-Verlag, 2001, ch. Models and Technologies for the Coordination of Internet Agents: A Survey.

[214] K. H. Park, Y. J. Kim, and J. H. Kim, "Modular q-learning based multi-agent cooperation for robot soccer," *Robotics and Autonomous Systems*, pp. 109–122, 2001.

[215] T. Parsons and E. A. Shils, Eds., *Toward a General Theory of Action*. Cambridge, Harvard University Press, 1951.

[216] H. V.-D. Parunak, "Manufacturing experience with the contract net," in *Distributed Artificial Intelligence*, L. Gasser and M. Huhns, Eds., vol. 2. Morgan Kaufmann, 1989.

[217] P. Peeters, H. V. Brussel, P. Valckenaers, J. Wyns, L. Bongaerts, M. Kollingbaum, and T. Heikkilä, "Pheromone based emergent shop floor control sys-

tem for flexible flow shops," *Artificial Intelligence in Engineering*, vol. 15, pp. 343–352, 2001.

[218] L. Peshkin, K. E. Kim, N. Meuleau, and L. P. Kaelbling, "Learning to cooperate via policy search," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.

[219] G. P. Picco, A. L. Murphy, and G. C. Roman, "Lime: Linda meets mobility," in *Proceedings of the 21th International Conference on Software Engineering (ICSE'99)*, D. Garlan and J. Kramer, Eds., 1999, pp. 368–377.

[220] D. G. Pruitt, *Negotiation Behavior*. Academic Press, 1981.

[221] M. L. Puterman, "Markov Decision Processes," in *Handbook in Operations research and Management Science*, D. P. Heyman and M. J. Sobel, Eds. North-Holand, 1990, vol. 2: Stochastic Models, ch. 8, pp. 331–434.

[222] D. V. Pynadath and M. Tambe, "The communicative multiagent team decision problem: Analyzing teamwork theories and models," *Journal of Artificial Intelligence Research*, pp. 389–423, 2002.

[223] R. J. Rabelo and L. M. Camarinha-Matos, "Negotiation in multi-agent based dynamic scheduling," *Robotic Computer-Integrated Manufacturing*, vol. 11, no. 4, pp. 303–309, 1994.

[224] Z. Rabinovich and J. S. Rosenschein, "Multiagent Coordination by Extended Markov Tracking," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005.

[225] A. S. Rao and M. P. Georgeff, "BDI Agents: From Theory to Practice," in *Proceedings of the First International Conference on Multiagent Systems*, 1995.

[226] A. Ricci, A. Omicini, and E. Denti, "Activity Theory as a framework for MAS coordination," in *Engineering Societies in the Agents World III*, P. Petta, R. Tolksdorf, and F. Zambonelli, Eds. Springer-Verlag, 2003, pp. 96–110.

[227] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, pp. 400–407, 1951.

[228] J. S. Rosenschein, "Rational interaction: Cooperation among intelligent agents," Ph.D. dissertation, Stanford University, 1985.

[229] J. S. Rosenschein and J. S. Breese, "Communication-Free Interactions among Rational Agents: A Probabilistic Approach," in *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, Eds., vol. II. Morgan Kaufmann, 1989, pp. 99–118.

[230] J. S. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, 1994.

[231] C. D. Rosin and R. K. Belew, "Methods for competitive co-evolution: Finding opponents worth beating," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, S. Forrest, Ed. Morgan Kaufman, 1995, pp. 373–380.

[232] D. Rossi, "A coordination language and a coordination architecture for internet computing," Ph.D. dissertation, Department of Computer Science, University of Bologna, 2000.

[233] D. Rossi, G. Cabri, and E. Denti, "Tuple-based technologies for coordination," in *Coordination of Internet Agents*, 2001.

[234] A. Rowstron, *Coordination of Internet Agents*. Springer-Verlag, 2001, ch. Run-Time Systems for Coordination.

[235] ——, "WCL: A Web Co-ordination Language," *World Wide Web Journal*, 1998.

[236] J. Rust, "Numerical dynamic programming in economics," in *Handbook of Computational Economics*, H. M. Amman, D. A. Kendrick, and J. Rust, Eds. Amsterdam, The Netherlands: Elsevier, Amsterdam, 1996, vol. 1, ch. 14.

[237] S. K. Rustogi and M. P. Singh, "Be Patient and Tolerate Imprecision: How Autonomous Agents can Coordinate Effectively," in *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, 1999.

[238] A. Saad, K. Kawamura, and G. Biswas, "Performance evaluation of contract net-based heterarchical scheduling for flexible manufacturing systems," *International Journal of Intelligent Automation and Soft Computing: Special Issue on Intelligent Manufacturing Planning and Shop Floor Control*, 1996.

[239] N. M. Sadeh, D. W. Hildum, D. Kjenstad, and A. Tseng, "MASCOT: An agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling," in *Third International Conference on Autonomous Agents Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain, Seattle WA*, 1999.

[240] M. K. Sahota, "Reactive deliberation: An architecture for real-time intelligent control in dynamic environments," in *Proceedings of the 12th National Conference on Artificial Intelligence*, 1994, pp. 1303–1308.

[241] T. Sandholm and V. Lesser, "Issues in automated negotiation and electronic commerce: Extending the contract net framework," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, pp. 328–335.

[242] T. W. Sandholm, "Distributed Rational Decision Making," in *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. The MIT Press, 1999, pp. 201–258.

[243] T. W. Sandholm and R. H. Crites, "On multiagent Q-learning in a semi-competitive domain," in *Adaptation and Learning in Multiagent Systems*, G. Weiss and S. Sen, Eds. Springer Verlag, 1996.

[244] A. Sathi and M. S. Fox, "Constraint-directed negotiation of resource reallocations," in *Distributed Artificial Intelligence*, vol. 2. Morgan Kaufmann, 1989, pp. 163–193.

[245] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, 1990.

[246] R. Schoonderwoerd, O. Holland, and J. Bruten, "Ant-like agents for load balancing in telecommunications networks," in *Proceedings of the First International Conference on Autonomous Agents (Agents '97)*, 1997, pp. 209–216.

[247] M. Schumacher, *Objective coordination in multi-agent system engineering: design and implementation.* Springer, 2001.

[248] J. R. Searle, *Speech Acts: an Essay in the Philosophy of Language.* Cambridge University Press, 1969.

[249] S. Sen and M. Sekaran, "Individual learning of coordination knowledge," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 10, pp. 333–356, 1998.

[250] Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, no. 1, pp. 51–92, 1993.

[251] Y. Shoham and M. Tennenholtz, "Emergent conventions in multi-agent systems," in *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, C. Rich, W. Swartout, and B. Nebel, Eds., 1992, pp. 225–231.

[252] ——, "On the synthesis of useful social laws for artificial agent societies," in *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, 1992, pp. 276–281.

[253] ——, "On social laws for artificial agent societies: offline design," in *Computational Theories of Interaction and Agency*, P. E. Agre and S. J. Rosenschein, Eds. MIT Press, 1996, pp. 597–618.

[254] ——, "On the emergence of social conventions: modelling, analysis and simulations," *Artificial Intelligence*, vol. 94, no. 1–2, pp. 139–166, 1997.

[255] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," in *IEEE Transactions on Computers*. IEEE Press, 1980, pp. 1104–1113.

[256] P. Stone, "Layered learning in multi-agent systems," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, 1998.

[257] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[258] T. Sugawara and V. Lesser, "Learning to improve coordinated actions in cooperative distributed problem-solving environments," *Machine Learning*, vol. 33, pp. 129–153, 1998.

[259] M. Sugeno, "An introductory survey of fuzzy control," *Information Science*, vol. 36, pp. 59–83, 1985.

[260] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[261] K. Sycara, "Multiagent compromise via negotiation," in *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, Eds., vol. 2. Morgan Kaufmann, 1989, pp. 119–137.

[262] ——, "Multiagent systems," *AI Magazine*, vol. 19, no. 2, pp. 79–92, 1998.

[263] K. Sycara, A. Pannu, M. Willamson, D. Zeng, and K. Decker, "Distributed intelligent agents," *IEEE Expert*, vol. 11, pp. 36–46, 1996.

[264] C. Szepesvári and M. L. Littman, "A Unified Analysis of Value-Function-Based Reinforcement Learning Algorithms," *Neural Computation*, vol. 11, no. 8, pp. 2017–2060, 1999.

[265] M. Tambe, "Towards flexible teamwork," *Journal of AI Research*, vol. 7, pp. 83–124, 1997.

[266] M. Tan, "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Learning," in *Readings in Agents*, M. N. Huhns and M. P. Singh, Eds. San Francisco, CA, USA: Morgan Kaufmann, 1997, pp. 487–494.

[267] A. ter Mors, J. Valk, and C. Witteveen, "Coordinating autonomous planners," in *Proceedings of the international conference on artificial intelligence*, 2004.

[268] J. Thangarajah, L. Padhgam, and M. Winikoff, "Detecting and avoiding interference between goals in intelligent agents," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[269] G. Theraulaz, E. Bonabeau, and J. L. Deneubourg, "Response threshold reinforcement and division of labour in insect societies," in *Proceedings of the Royal Society of London B*, vol. 265, no. 1393, 1998, pp. 327–335.

[270] G. Theraulaz, S. Goss, J. Gervet, and J. L. Deneubourg, "Task differentiation in polistes wasp colonies: A model for self-organizing groups of robots," in *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press, 1991, pp. 346–355.

[271] G. Thompson, J. Frances, R. Levacic, and J. Mitchell, Eds., *Markets Hierarchies and Networks: The Coordination of Social Life*. SAGE Publications, 1991.

[272] F. Tohme, "Negotiation and defeasible reasons for choices," in *Proc. AAAI Spring Symposium on Qualitative preferences in deliberation and practical reasoning*, 1997, pp. 95–102.

[273] R. Tolksdorf, "Laura: A coordination language for open distributed systems," Technische Universität Berlin, Fachbereich 20 Informatik, Tech. Rep., 1992.

[274] C. F. Touzet, "Robot awareness in cooperative mobile robot learning," *Autonomous Robots*, pp. 87–97, 2000.

[275] ——, "Distributed lazy q-learning for cooperative mobile robots," *International Journal of Advanced Robotic Systems*, pp. 5–13, 2004.

[276] S. van der Zwaan and C. Marques, "Ant colony optimisation for job shop scheduling," in *Proceedings of the '99 Workshop on Genetic Algorithms and Artificial Life (GAAL '99)*, 1999.

[277] D. Vengerov and H. R. Berenji, "Adaptive coordination among fuzzy reinforcement learning agents performing distributed dynamic load balancing," in *Proceedings of the 11th IEEE International Conference on Fuzzy Systems*, 2002.

[278] A. Walker and M. Wooldridge, "Understanding the emergence of conventions in multi-agent systems," in *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, pp. 384–390.

[279] W. E. Walsh and M. P. Wellman, "A market protocol for decentralized task allocation," in *Proceedings of the 3rd International Conference on Multi-Agent Systems*, 1998, pp. 325–332.

[280] ——, "Modeling Supply Chain Formation in Multiagent Systems," in *Lecture Notes in Artificial Intelligence: Agent Mediated Electronic Commerce II*, vol. 1788. Springer-Verlag, 2000.

[281] ——, "Decentralized supply chain formation: A market protocol and competitive equilibrium analysis," *Journal of Artificial Intelligence Research*, pp. 513–567, 2003.

[282] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason, "Some economics of market-based distributed scheduling," in *Proceedings of the 18th International Conference on Distributed Computing Systems*, 1998, pp. 612–621.

[283] A. Wang, "Using Javaspaces to Implement a Mobile Multi-Agent System," in *Twentieth IASTED International Conference on Applied Informatics (AI 2002)*, 2002.

[284] R. Washington, "Markov Tracking for Agent Coordination," in *Proceedings of the second international conference on Autonomous agents*, 1998.

[285] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[286] G. Weiss, "Learning to coordinate actions in multi-agent systems," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993, pp. 311–316.

[287] G. Weiss, Ed., *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[288] D. J. White, *Markov Decision Processes*. John Wiley & Sons, 1993.

[289] O. E. Williamson, *Markets and Hierarchies, Analysis and Antitrust*. New York: Free Press, 1975.

[290] E. O. Wilson, "The relation between caste ratios and division of labour in the ant genus pheidole (hymenoptera: Formicidae)," *Behavioral Ecology and Sociobiology*, pp. 89–98, 1984.

[291] M. Wooldridge, "Verifiable semantics for agent communication languages," in *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS-98)*, 1998, pp. 349–365.

[292] ——, *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

[293] M. Wooldridge and N. R. Jennings, "Formalizing the cooperative problem solving process," in *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (IWDAI-94)*, 1994, pp. 403–417.

[294] P. Xuan, V. Lesser, and S. Zilberstein, "Communication in Multi-agent Markov Decision Processes," in *Proceedings of the Fourth International Conference on MultiAgent Systems*, 2000.

[295] L. A. Zadeh, *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh*, ser. R. R. Yage and R. R. Yager and R. M. Ton(Eds.). John Wiley & Sons, 1987.

[296] D. Zeng and K. Sycara, "How can an agent learn to negotiate?" in *Intelligent Agents III*, J. Muller, M. Wooldridge, and N. R. Jennings, Eds. Springer Verlag, 1997, pp. 233–244.

[297] ——, "Bayesian learning in negotiation," in *Int. J. Human-Computer Studies*, 1998, pp. 125–141.

[298] G. Zlotkin and J. S. Rosenschein, "Mechanism design for automated negotiation, and its application to task oriented domains," *Artificial Intelligence*, vol. 86, pp. 195–244, 1996.