Autonomous Agents and Multi-Agent Systems, 10, 273–328, 2005

Coordinating Multiple Agents via Reinforcement Learning

GANG CHEN AND ZHONGHUA YANG PG02463664@ntu.edu.sg
ezhyang@ntu.edu.sg

Information Communication Institute of Singapore, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798

HAO HE AND KIAH MOK GOH hhe@simtech.a-star.edu.sg
kmgoh@simtech.a-star.edu.sg

Singapore Institute of Manufacturing Technology, Singapore 638075

Abstract. In this paper, we attempt to use reinforcement learning techniques to solve agent coordination problems in task-oriented environments. The Fuzzy Subjective Task Structure model (FSTS) is presented to model the general agent coordination. We show that an agent coordination problem modeled in FSTS is a Decision-Theoretic Planning (DTP) problem, to which reinforcement learning can be applied. Two learning algorithms, "*coarse-grained*" and "*fine-grained*", are proposed to address agents coordination behavior at two different levels. The "*coarse-grained*" algorithm operates at one level and tackle *hard* system constraints, and the "*fine-grained*" at another level and for soft constraints. We argue that it is important to explicitly model and explore coordination-specific (particularly system constraints) information, which underpins the two algorithms and attributes to the effectiveness of the algorithms. The algorithms are formally proved to converge and experimentally shown to be effective.

Keywords: multiagent system, coordination, reinforcement learning, task-oriented Environment.

1. Introduction

One of the fundamental issues in autonomous multi-agent systems (MAS) is the effective coordination among agents. In a nutshell, the coordination problem is that of *managing inter-dependencies* between the activities of agents [28]. Effective coordination can improve the agents' probabilities of satisfying given system goals and constraints, and thus the probability of achieving the overall design objectives. The system goals and constraints can take various forms depending on the particular application domains. In this paper, we address the coordination issue within a *cooperative* task-oriented environment [18], where multiple agents cooperate to fulfill the system goals by performing a series of tasks. The existence of constraints in such an environment affects the potential task fulfillment behaviors of each agent. We classify the constraints 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 fulfillments, but not necessarily results in a failure. The interdependencies among agents are derived from the constraints imposed on the on-going tasks.

We present the *Fuzzy Subjective Task Structure* model (FSTS) to abstract the coordination problems with the essential notions such as *methods*, *tasks*, and *method* *relations*. Fuzzy logic techniques [43] are explored in the model to capture the uncertainties within the descriptions of the task-oriented environment. From the FSTS model, we show that the agent coordination is approached as a *decision-theoretic planning* (DTP) problem [10], and the agent coordination process is a *Markov Decision process*. However, the traditional dynamic programming technique [6] is not directly applicable in this setting for two major reasons. (1) The actions performed by agents may have nondeterministic consequences with unknown distributions. (2) The *utilities* of the actions being performed by each agent, i.e., the *reward* of any system state changes, may not be deterministic and known *a priori*. In this paper, we attempt to explore agents' decision-making via reinforcement learning. We note that reinforcement learning is widely utilized as an effective model-free approach to dynamic programming problems [26].

Two learning algorithms, "*coarse grained*" and "*fine grained*", are proposed to address the *hard* and *soft* system constraints, respectively. 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.

Our approach contributes to the previous learning-based coordination techniques (e.g., in [5, 39]) in the following two aspects. First, it is designed for general task-oriented environments in which the system constraints dynamically determine the interdependencies among agents [20]. The constraints are explicitly modeled in FSTS 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 [17]. 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.

In our approach, the "*abstract system states*", a new set of system states, are obtained from the concrete system states by *state aggregation*. Because of the *curse of dimensionality* [4], it is necessary to adopt certain approximation methods for practical learning applications [31]. 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 [10]. When designing our 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. We show that in the deterministic environment, our first three assumptions are analogous to but less stringent than the notion of equivalence identified in [22], while the fourth assumption can often be omitted practically. In this paper, we present a bound on the difference between the learning result obtained from the abstract system and the one from the concrete system. 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.

The remainder of this paper is organized as follows. In Section 2, the FSTS modeling framework is presented, and the agent coordination in task-oriented environments is shown to be a decision-theoretic planning problem. Section 3 is devoted to the two learning algorithms: the coarse-grained learning algorithm and the fine-grained algorithm. The assumptions are made as required. The theoretical analysis is presented in Section 4. The experimental results for a multi-agent Pathology Lab system are discussed in Section 5. A discussion on the related work is given in Section 6. Finally Section 7 concludes the paper and comments on future research directions.

2. Modeling agent coordination

We regard agent coordination as a series of decision-making processes in uncertain environments. This view is inspired by the extensive research in AI in DTP. DTP is considered as a general paradigm for planning and decision-making problems in uncertain settings [10], and has as its root a model called Markov Decision Process (MDP) [3, 29, 42]. In this section, we propose a FSTS to model agent coordination problems, and then show that an agent coordination problem is a DTP problem where the Markov property holds. The problem is then solved accordingly by the techniques such as reinforcement learning or dynamic programming.

2.1. Fuzzy Subjective Task Structure Model (FSTS)

The FSTS is proposed to model agent coordination problems in task-oriented environments [14]. 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 $\mathscr{A} = \{A_1, A_2, \dots, A_n\}$, coordinate their behaviors in order to fulfill their tasks, denoted by $\mathscr{T} = \{T_1, T_2, \dots, T_m\}$. Since each task $T_k \in \mathscr{T}$ 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. $\epsilon : V \to L_V$ 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;

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

We distinguish a hard relation graph from a soft relation graph. A hard relation graph represents hard constraints only and a soft relation graph contains merely soft relations.[1] 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 $M \in \overline{M}^u$.

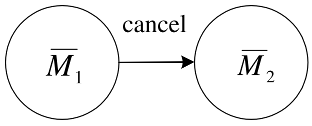

Figure 1. The cancel relation between meta-methods $\overline{M}_1$ and $\overline{M}_2$.

We illustrate the Cancel rule in Figure 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 1, we use Equation (1) to evaluate the degree to which this cancel relation may be violated. The operator $\bigwedge$ is the standard *T-norm operator* [43].

$$\mu_{\overline{M}_1}(M) \bigwedge \mu_{\overline{M}_2}(M') \tag{1}$$

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* [18]. 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: $V \times V \to (x, y)$; 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), we will postpone its discussion until we describe our learning algorithms (Section 3.4).

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, $Seq = (Seq_1, Seq_2, \ldots, Seq_n)$. By following *Seq*, the tasks in $\mathscr{T}$ 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* [17].

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.

2.2. 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 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.

Now we formally address these four aspects.

2.2.1. 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.

Let S^t denote the system state at time t and $\mathscr{S}$ 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 $\mathscr{H}$. For any history h, the *Markov property* requires,

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

The system states enable a *factored representation* if the state space is characterized by a set of *variables* or *fluents* $\mathscr{F} = \{F_1, \ldots, F_l\}$. 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. We explicitly list all the tasks from T_1 to T_m in $\mathscr{T}$. We assume that each task T_k contains at most q meta-methods. We divide the l fluents into m groups. $\mathscr{F}$ is consequently represented as an ordered list $(\mathscr{F}_1, \mathscr{F}_2, \ldots, \mathscr{F}_m)$, where $\mathscr{F}_k$ 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 $\mathscr{F}_k$ 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 $\mathscr{T}$, this factored representation of system states maintains the required *Markov property*.

We 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 $\mathscr{F}$, the *state space* $\mathscr{S}$ is defined as the set of all valid combinations of fluent values. Every such combination is represented by a list $(f_1, \ldots, f_l)$. It represents uniquely a state $S \in \mathscr{S}$.

2.2.2. 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, $\pi : \mathscr{S} \rightarrow \mathscr{M}$. We add one special method M^0 to $\mathscr{M}$ 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.

*2.2.3. **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 (2),

$$Pr(h \mid \pi) = \prod_{i=0}^{l} [(P)^i \cdot \vec{p}^0]_{S^{t_i}} \tag{2}$$

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 (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 $\mathscr{T}$. 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 $T \in \mathscr{T}$ 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, we may not be able 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. We therefore depend the agents' decision-makings upon the reinforcement learning, which is widely accepted as effective *model-free* alternatives for dynamic programming problems. We emphasize 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. Specifically, we take into account the following two observations:

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).

2.2.4. 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 $\mathscr{H}$ into real number. The function is assumed to be *time-separable* [11]. 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* $R : \mathscr{S} \times \mathscr{S} \rightarrow \Re$ 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. We therefore extend each history h into an infinite-length list by repeatedly appending the last system state of h to itself. $V(h)$ is consequently re-defined as

$$V(h) = \sum_{i=0}^{\infty} \alpha(t_{i+1}) \cdot R(S^{t_i}, S^{t_{i+1}}) \tag{3}$$

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

$$EV(\pi) = \sum_{h \in \mathscr{H}} V(h) \cdot Pr(h \mid \pi) \tag{4}$$

where $Pr(h \mid \pi)$ is defined in Equation (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 (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. Agent coordination through reinforcement learning

As mentioned in Section 2, the agent coordination problems differ from the common DTP problems in that the stochastic model of system transitions might not be known *a priori*. It motivates us to approach agents' decision-making by using the reinforcement learning methods. In the remaining part of this section, we will first describe the assumptions on which our two learning algorithms are based, followed by a high-level overview of the algorithms. The two algorithms are presented in detail at the end of the section.

3.1. Learning Objectives and Basic Assumptions

As described in Section 2, 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, we define the expected value of the policy π at state S as

$$V^{\pi}(S) = \sum_{S' \in \mathscr{S}} \{Pr(S' \mid \pi(S), S) \cdot \alpha_{S'} \cdot [R(S, S') + V^{\pi}(S')]\} \tag{5}$$

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 (5), the method to perform on state S can be indirectly determined through Equation (6):

$$\pi(S) = \underset{M \in \mathscr{M}}{\arg\max} \left(\textstyle\sum_{S' \in \mathscr{S}} \{Pr(S' \mid M, S) \cdot \alpha_{S'} \cdot [R(S, S') + V(S')]\} \right) \tag{6}$$

where $V(S)$ is the expected value of state S. Apparently, for any mapping $V : \mathscr{S} \to \Re$, there exists a corresponding policy π that satisfy Equation (6). Constructing the function $V(S)$ for every $S \in \mathscr{S}$, 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, we present the quality function 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 (7):

$$\pi(S) = \underset{M \in \mathscr{M}}{\arg\max} \left\{ \sum_{S' \in \mathscr{S}} \left[Pr(S' \mid M, S) \cdot Q(S, S', M) \right] \right\} \tag{7}$$

Equation (7) 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 (7) indicates that its use requires agents to identify *a priori* the stochastic model of system transitions. To address this issue, the following Equation (8) is used to decide the method to be performed at each system state.

$$\pi(S) = \underset{M \in \mathscr{M}}{\arg\max} \left\{ \underset{S' \in \mathscr{S}'_{S,M}}{\text{Avg}} [Q(S, S', M)] \right\} \tag{8}$$

where $\mathscr{S}'_{S,M}$ is a specific set of system states, $\{S' \mid Pr(S' \mid M, S) > 0\}$. Following Equation (8), 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 (6) or (7)) were also explored in the literature, such as the risk-sensitive reinforcement learning algorithm [24]. Taking into account Equation (8), the learning objectives of our learning algorithms are to identify a quality function $Q^*(S, S', M)$ that satisfies Equation (9):

$$Q^*(S_1, S_2, M) = R(S_1, S_2) + \alpha_{S_2} \cdot \underset{(S_3, M') \in \mathscr{S}_{S_2}}{\text{Avg}} (Q^*(S_2, S_3, M')) \tag{9}$$

where $\mathscr{S}_{S_2}$ refers to a set of state-method pairs, such that for every pair (S_3, M') in the set, $Pr(S_3 \mid M', S_2) > 0$. We characterize the methods 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 $\mathscr{S}$ and $\mathscr{M}$.

In general, a partition E of the set $\mathscr{S} = \{S_1, \ldots, S_n\}$ is a set of sets $\{\mathscr{B}_1, \ldots, \mathscr{B}_r\}$ such that each $\mathscr{B}_i$ is a subset of $\mathscr{S}$, $\mathscr{B}_i$ are disjoint from one another, and the union of all $\mathscr{B}_i$ returns the set $\mathscr{S}$. We call each member of a partition a *block*. In the succeeding discussion, each block of system states is denoted as $\mathscr{B}^S$ and each block of domain methods is denoted as $\mathscr{B}^M$. 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:

$$Q^*(B^S{}_1, B^S{}_2, B^M) = \underset{S_1 \in B^S{}_1, S_2 \in B^S{}_2, M \in B^M, Pr(S_2 | M, S_1) > 0}{\text{Avg}} (Q^*(S_1, S_2, M)) \tag{10}$$

In Equation (10), 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 $Q^*(B^S{}_1,$

${B^S}_2, B^M$). We note that the use of the average operation in estimating the expected value at each system state is also recommended by some DTP researchers [10]. With Equation (10), the method selected at each system state S is to satisfy Equation (11) as follows:

$$\pi(S) = \underset{M \in \mathscr{M}}{\arg\max} \left\{ \underset{\substack{\text{for all } S' \in \mathscr{S}'_{S,M} \\ S \in B^S, S' \in B^{S'}, M \in B^M}}{\text{Avg}} [Q^*(B^S, B^{S'}, B^M)] \right\} \tag{11}$$

Equation (11) follows Equation (8) 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, we choose to avoid it by making a few assumptions regarding our partition of sets $\mathscr{S}$ and $\mathscr{M}$. We regulate that the partition is 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 our discussion, we 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 our algorithms are designed to handle the violation of A^*. We now present the four assumptions regarding the system partitions:

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

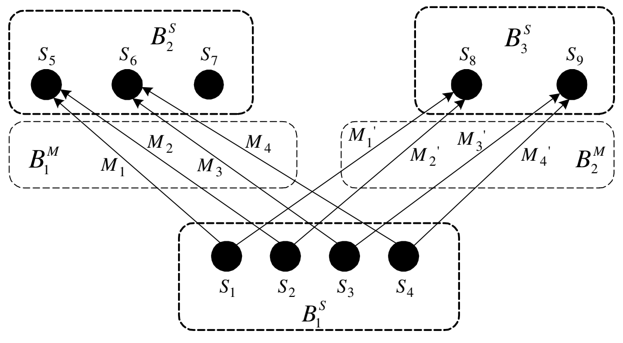

Figure 2. An example partition of the system states and methods that satisfies assumptions A1–A3.

$B^{S'}$, and one method block B^M, $B^S \overset{B^M}{\rightarrow} B^{S'}$ 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 $B^{S'}$ (Figure 2).

Figure 2 shows a partition of three state blocks and two domain method blocks. There are two groups of system transitions, denoted as ${B^S}_1 \overset{{B^M}_1}{\rightarrow} {B^S}_2$ and ${B^S}_1 \overset{{B^M}_2}{\rightarrow} {B^S}_3$, 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 (11). We say that a triplet of blocks is $(B^S, B^{S'}, B^M)$ applicable if $B^S \overset{B^M}{\rightarrow} B^{S'}$. We 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

$$\underset{\substack{\text{for all } B^{S'} \\ S \in B^S, M \in B^M}}{\text{Avg}} [Q^*(B^S, B^{S'}, B^M)] \tag{12}$$

From Equation (12), 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 (11), 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 [22]. 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 we consider each method block B^M as an individual domain method, assumptions A2 and A3 are actually identical with B2. 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 3. Nevertheless, the violation of A4 does not affect the applicability of Equation (12).

Readers may notice the similarity between Figures 2 and 3. The major difference 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 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. We also provide two error bounds 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 [17]. These error bounds validate the importance of assumption A4, which also helps when describing our 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

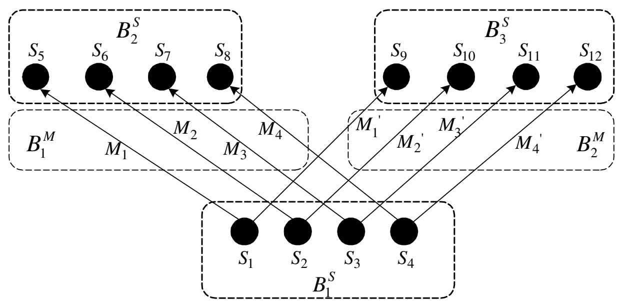

Figure 3. An example partition of the system states and methods that satisfies assumptions A1–A4.

defined as the maximum range of possible rewards:

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

The maximum reward span for a partition E is then

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

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, we have*

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

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, we estimate the quality function $Q^*(S, S', M)$ defined in Equation (9) through a sequence of quality functions Q^k, that is

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

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

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

We prove this theorem 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 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

$$\begin{aligned}&\left|Q^{k+1}(S,S',M) - Q^{k+1}(B^S,B^{S'},B^M)\right|\\ &\quad= \left|\left[R(S,S',M) + \alpha_{S'} \cdot \underset{(S'',M')}{\text{Avg}}\,(Q^k(S',S'',M'))\right]\right.\\ &\quad\left.\times\left[\underset{(S,S',M)}{\text{Avg}}\left(Q^k(S,S',M)\right)\right]\right|\end{aligned} \tag{13}$$

In Equation (13), the average operation over all $Q^k(S,S',M)$ such that $S \in B^S$, $S' \in B^{S'}$, $M \in B^M$, and $S \overset{M}{\rightarrow} S'$ can be written as

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

where 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

$$\underset{S'\in B^{S'}}{\text{Avg}}\,\alpha_{S'} \cdot \left[\underset{(S'',M')}{\text{Avg}}\left(Q^k(S',S'',M')\right)\right] = \alpha_{S'} \cdot \underset{(B^{S''},B^{M'})}{\text{Avg}}\left(Q^k(B^{S'},B^{S''},B^{M'})\right) \tag{14}$$

With these transformations, we now resume our analysis of Equation (13). Specifically,

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

$$
\begin{aligned}
&\leq \delta + \alpha \cdot \sum_{i=0}^{k} \delta \cdot \alpha^i \\
&= \sum_{i=0}^{k+1} \delta \alpha^i
\end{aligned} \tag{15}
$$

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 (15), the theorem is thus proved. □

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 2, 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 (14). In order to maintain a bounded difference between the quality functions $Q(S, S', M)$ and $Q(B^S, B^{S'}, B^M)$, we assume that we have certain information about the quality functions, specifically,

$$
\max \left\{ \left| \begin{array}{l} \underset{(S,S',M)}{\operatorname{Avg}} \left[\underset{(S'',M')}{\operatorname{Avg}} (Q(S', S'', M')) \right] \\ - \underset{S' \in B^{S'}}{\operatorname{Avg}} \left[\underset{(S'',M')}{\operatorname{Avg}} (Q(S', S'', M')) \right] \end{array} \right| \right\} \leq \Delta \tag{16}
$$

Equation (16) 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, we now give the error bound for the case where the partition violates assumption A4.

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 (16) is valid with finite* Δ, *we have*

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

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, we focus 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 $k = 0$. We now consider the inductive step as follows:

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

Further expanding the last term of the above inequality, we have

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

With the above inequality, we conclude the inductive step by showing that

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

By taking the above inequality in the limit that $k \to \infty$, the result of the theorem yields. □

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 we stated in Section 4, 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. Our experiments described in Section 5 also demonstrate these ideas and show that our learning algorithms work well without respecting assumption A4.

3.2. *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 [35]. The coarse-grained learning algorithm is devised for managing *hard constraints*. Its general structure is illustrated in Figure 4.

With this structure (Figure 4), the coarse-grained learning algorithm can insert, modify, or delete data items from a table of the database, which is used 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

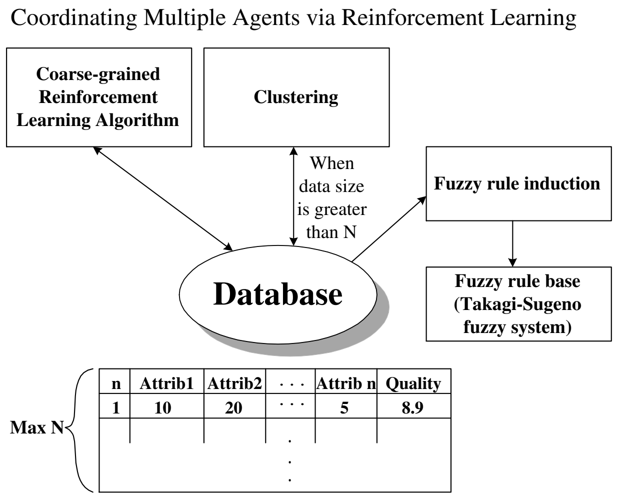
Coordinating Multiple Agents via Reinforcement Learning
Coarse-grained Reinforcement Learning Algorithm
Clustering
When data size is greater than N
Fuzzy rule induction
Database
Fuzzy rule base (Takagi-Sugeno fuzzy system)
Max N
n | Attrib1 | Attrib2 | . . . | Attrib n | Quality
1 | 10 | 20 | . . . | 5 | 8.9

Figure 4. The general structure of the first learning algorithm.

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, we therefore term it 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 5. 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, we follow a simple approach, that is, (1) during the learning process, the method most probably selected to perform is the one with the highest average quality; (2) during the system execution, the method with the highest average quality is selected with probability 1.

3.3. Coarse-grained Learning Algorithm

In the coarse-grained learning algorithm, the partition fluents of each row of the table are divided into three groups (Figure 4), denoted by D_s, D'_s, and A_f respectively. D_s refers to the set of fluents characterizing the current system state S.

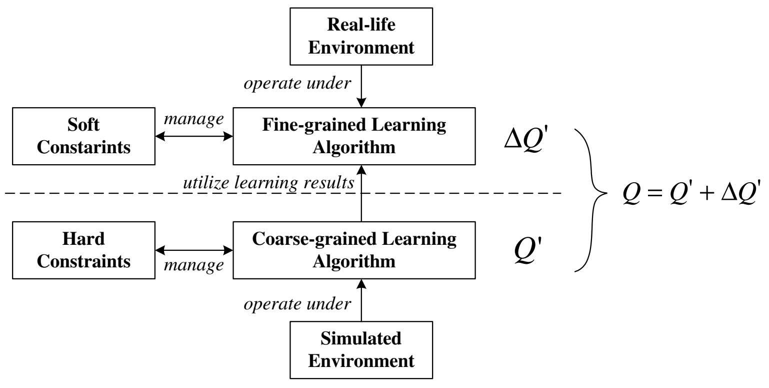

Figure 5. The relations between the two learning algorithms.

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 (10). The quality of performing each method can be written as:

$$Q(S, S', M) = R(S, S') + \alpha \cdot \underset{(S'',M')}{\text{Avg}} (Q(S', S'', M')) \tag{17}$$

To simplify the description, Equation (17) has omitted the differences among the discount rates at the various system states. We denote the discount rate uniformly by α. We assume initially that the application domain satisfies assumption A^*, and our system partition E satisfies assumptions A1–A4. As indicated earlier, assumption A4 can be omitted in practice. Later, we show how the coarse-grained learning algorithm handles 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 $S \xrightarrow{M} S'$. We expand the definition of the aggregated quality function as

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

where

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

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

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

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

$$\eta(B^{S'}) = \frac{1}{\varphi(B^{S'})} \cdot \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \quad (18)$$

We now obtain the equation for aggregated quality functions as

$$\begin{aligned} &Q^*(B^S, B^{S'}, B^M) \\ &\quad = \underset{(S,S',M)\in\Psi(B^S,B^{S'},B^M)}{\text{Avg}} R(S, S') \\ &\qquad + \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \\ &\qquad \times \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q^*(B^{S'}, B^{S''}, B^{M'}) \end{aligned} \quad (19)$$

Equation (19) is the *Bellman equation* [4] for our coarse-grained learning problem. We consequently construct the updating rule of the coarse-grained learning algorithm as

$$\begin{aligned} Q(B^S, B^{S'}, B^M) &\leftarrow (1 - \frac{1}{n(B^S, B^{S'}, B^M)}) \cdot Q(B^S, B^{S'}, B^M) \\ &+ \frac{1}{n(B^S, B^{S'}, B^M)} \left[R(S, S') + \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \cdot \right. \\ &\left. \sum_{(B^{S'},B^{S''},B^{M'})\in\omega(B^{S'})} \left(\left\|\Psi(B^{S'}, B^{S''}, B^{M'})\right\| \cdot Q(B^{S'}, B^{S''}, B^{M'})\right)\right] \end{aligned} \quad (20)$$

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 (20)). 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, we re-define the *quality* of performing a method M as

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

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 our previous discussions, we can now obtain the updating rule for the modified coarse-grained learning algorithm as follows:

$$\begin{aligned} Q(B^S, B^{S'}, B^M) \leftarrow {} & (1-\gamma_t)\cdot Q(B^S, B^{S'}, B^M) + \gamma_t \\ & \times R(S, S'') + \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S''})\cdot \left\|\Psi(B^S, B^{S'}, B^M)\right\|} \\ & \times \sum_{(B^{S''}, B^{S'''}, B^{M'})\in\omega(B^{S''})} \left\|\Psi(B^{S''}, B^{S'''}, B^{M'})\right\| Q(B^{S''}, B^{S'''}, B^{M'}) \end{aligned} \tag{21}$$

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, we further simplify our updating rule in Equation (21) as follows:

$$\begin{aligned} Q(B^S, B^{S'}, B^M) \leftarrow {} & (1-\gamma_t)\cdot Q(B^S, B^{S'}, B^M) + \gamma_t \cdot \\ & R(S, S'') + \gamma_t \cdot \alpha \cdot \frac{\sum\limits_{(B^{S''}, B^{S'''}, B^{M'})\in\omega(B^{S''})} Q(B^{S''}, B^{S'''}, B^{M'})}{\left\|\omega(B^{S''})\right\|} \end{aligned}$$

This *is* the updating rule adopted in our coarse-grained learning algorithm. We present the algorithm in procedural form in Figure 6. 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 $\left\|\Psi(B^S, B^{S'}, B^M)\right\|$

```
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 6. The coarse-grained learning algorithm.

and $\varphi(B^S)$. Nevertheless, as we will show in the next Section, the algorithm can still converge towards a sub-optimal learning result. We believe that 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.

3.4. 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, we will construct for M a so-called *method relation graph*, G_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 7. 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) [12].

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

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 7. The procedure for constructing the method relation graph G_M of a method M.

multiple method blocks in the refined partition. However, as soft constraints take a significant role in the fine-grained learning algorithm, we represent the fluent group S_g 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

$$\Delta Q^*(S, S', M, Is_g) \\ = \operatorname*{Avg}_{G_M \in Is_g} \left(Q^*(S, S', M, G_M) - Q^*(S, S', \hat{M}) \right) \tag{23}$$

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

$$\Delta Q^*(B^S, B^{S'}, B^M, Is_g) \\ = Q^*(B^S, B^{S'}, B^M, Is_g) - Q^*(B^S, B^{S'}, B^M) \tag{24}$$

where

$$Q^*(B^S, B^{S'}, B^M, Is_g) \\ = \operatorname*{Avg}_{G_M \in Is_g} \operatorname*{Avg}_{(S,S',M) \in \Psi(B^S, B^{S'}, B^M, G_M)} Q^*(S, S', M, G_M) \tag{25}$$

$\Psi(B^S, B^{S'}, B^M, G_M)$ denotes the set of tuples (S, S', M, G_M) such that (1) the condition $S \xrightarrow{M} S'$ is valid; (2) the *soft constraints* relevant to the method M is characterized by G_M. We assume that $Q^*(B^S, B^{S'}, B^M)$, which is estimated by our 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. We first consider 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 [15, 30], the corresponding updating rule for estimating Q^* is

$$Q(B^S, B^{S'}, B^M, Is_g) \leftarrow (1 - \gamma_t) \cdot Q(B^S, B^{S'}, B^M, Is_g) \\ + \gamma_t \cdot Q^*(S, S', M, G_M) \tag{26}$$

We 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. We define the *temporal difference* $d_{S,S'}$ to be

$$d_{S,S'} = R(S,S') + \alpha \cdot \underset{(S',S'',M',G'_M)}{\text{Avg}} \left(Q(S',S'',M',G'_M) \right) - Q(S,S',M,G_M)$$

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

$$Q(S,S',M,G_M) \leftarrow Q(S,S',M,G_M) + \gamma_m \times Z_m(S,S',M,G_M) \cdot d_{S_m,S_{m+1}}$$

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

$$Z_m(S,S',M,G_M) = \begin{cases} \alpha\lambda Z_{m-1}(S,S',M,G_M) & (S_m,S_{m+1},M_m,G_{Mm}) \neq (S,S',M,G_M) \\ \alpha\lambda Z_{m-1}(S,S',M,G_M) + 1, & (S_m,S_{m+1},M_m,G_{Mm}) = (S,S',M,G_M) \end{cases} \quad (29)$$

By extending this algorithm for the aggregated systems, and incorporating it into Equation (26), we obtain the updating rule for estimating $Q^*(B^S,B^{S'},B^M,Is_g)$ as follows:

$$Q(B^S,B^{S'},B^M,Is_g) \leftarrow Q(B^S,B^{S'},B^M,Is_g) + \gamma_m \cdot Z_m(B^S,B^{S'},B^M,Is_g) \cdot d_{S_m,S_{m+1}} \quad (30)$$

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*, we obtain the corresponding updating rule to estimate ΔQ^* as

$$\Delta Q(B^S,B^{S'},B^M,Is_g) \leftarrow \Delta Q(B^S,B^{S'},B^M,Is_g) + \gamma_m \cdot Z_m(B^S,B^{S'},B^M,Is_g) \cdot d_{S_m,S_{m+1}} \quad (31)$$

For the aggregated systems, the evaluation of the temporal difference $d_{S_m,S_{m+1}}$ in Equation (27) is slightly altered as follows:

$$d_{S_m,S_{m+1}} = R(S_m,S_{m+1}) + \alpha \times \underset{(B^{S^{m+2}},B^{M^{m+1}},{Is_g}^{m+1})}{\text{Avg}} \left(Q(B^{S^{m+1}},B^{S^{m+2}},B^{M^{m+1}},{Is_g}^{m+1}) \right) - Q(B^{S^m},B^{S^{m+1}},B^{M^m},{Is_g}^m) \quad (32)$$

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 (31). However, the actual form needs to be modified in order to work efficiently with our 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. We use b_k to denote the output of the k-th fuzzy rule. Let

$$\phi_k(S, S', M, G_M) = \frac{\mu_k(S, S', M, G_M)}{\sum_{i=1}^{N} \mu_i(S, S', M, G_M)} \tag{33}$$

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

$$\Delta Q(B^S, B^{S'}, B^M, Is_g) = \sum_{k=1}^{N} \phi_k(S, S', M, G_M) \cdot b_k \tag{34}$$

where (S, S', M, G_M) can be any tuple that belongs to $\Psi(B^S, B^{S'}, B^M, Is_g)$. Equation (34) 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 our 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 [7], $\vec{b}$ can be actually updated through

$$\vec{b} \leftarrow \vec{b} + \gamma_m \cdot Z_m \cdot d_{S_m, S_{m+1}} \tag{35}$$

where Z_m is defined as

$$Z_m = \sum_{i=1}^{m} (\alpha \cdot \lambda)^{m-1} \cdot \phi(S_i, S_{i+1}, M_i, G_{Mi}) \tag{36}$$

This *is* the updating rule that is used in our 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, we propose *two* learning strategies.

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 [7], this strategy improves only once the decision policies which are determined by the coarse-grained learning algorithm. Adopting this strategy, the convergence of our fine-grained learning algorithm is ensured (Section 4). 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 [7], 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.

While describing the restricted policy iteration, we focus only on the general behavior of the algorithm at each learning phase, and omit the details of the fine-grained learning algorithm. This learning strategy is shown in Figure 8. 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 ${}^2\vec{b}$.

During each learning phase, the updating process follows Equation (35) 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

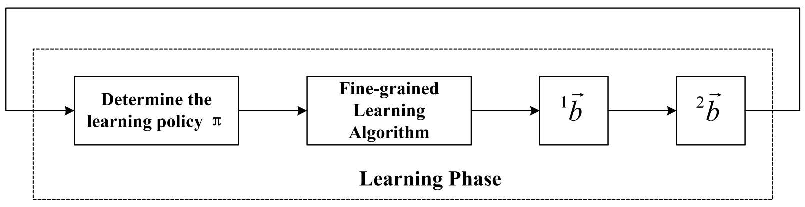
Determine the learning policy π
Fine-grained Learning Algorithm
$^1\vec{b}$
$^2\vec{b}$
Learning Phase

Figure 8. The restricted policy iteration process.

procedure that derives ${}^{2}\vec{b}$ from ${}^{1}\vec{b}$ demonstrates the above two aspects that distinguish our restricted policy iteration from the optimistic policy iterations. We propose several regulations to make the two aspects explicit. 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, we define a *trapping function trap*$(\vec{b})$ that maps 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 (37).

$$\forall b_i \; of \; \vec{b}, b'_i = \frac{b_i}{\left\|\vec{b}\right\|} \cdot \mathrm{trap}^*\left(\left\|\vec{b}\right\|\right) \tag{37}$$

where $\left\|\vec{b}\right\|$ can be any vector norm of $\vec{b}$, and the function *trap**(•) is a one-dimensional real function (Figure 9).

In Figure 9, v 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 v, the trapping function maps $\vec{b}$ to itself.

Second, we introduce a *step-weight* δ, which adjusts the differences between the vectors ${}^{2}\vec{b}$ of any two consecutive learning phases. 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 ${}^{2}\vec{b}'$, we may re-write the H function as $H({}^{2}\vec{b}')$. In order to ensure that the differences between ${}^{2}\vec{b}'$ and ${}^{2}\vec{b}$ are decreasing in

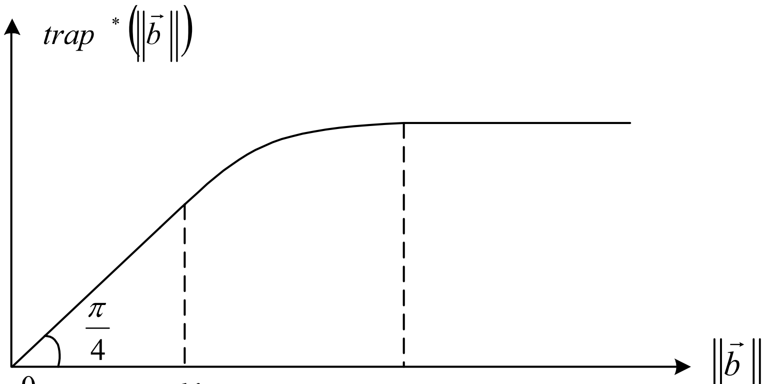

Figure 9. The general shape of the function trap*(•).

general, we adopt an updating rule given in Equation (38), which is frequently used in stochastic approximation algorithms, to determine the vector ${}^2\vec{b}$ at each learning phase.

$$
{}^2\vec{b} \leftarrow \operatorname{trap}\Big((1-\gamma_t) \cdot {}^2\vec{b} + \gamma_t \cdot H({}^2\vec{b})\Big) \tag{38}
$$

Note that γ_t in Equation (38) is a positive real number and stands for a phase-level learning rate of our 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 Section 4 where we show that by choosing a proper step-weight δ, our fine-grained learning algorithm converges under our restricted policy iterations.

4. Theoretical analysis of the learning algorithms

This section is dedicated to the theoretical analysis focusing on the convergence of the learning algorithms described in the previous section. 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 (20) or Equation (22) 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. 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, we introduce the two theorems from [37], which form the basis of the proof of the first convergence result. Let $T\colon B \rightarrow B$ 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 $T_t\colon B \times B \rightarrow B$. Let $F \subseteq B$ be a subset of B and let $F_0\colon F \rightarrow 2^B$ 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 we say that Γ approximates T for initial values from $F_0(f)$ and on the set $F \subseteq B$. Further, the subset $F \subseteq B$ is invariant under $T'\colon B \times B \rightarrow B$ 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 [37] are presented as Theorems 3 and 4. Note that there are some deviations in the meaning for symbols from that used in Section 2, which we will clarify when a confusion arises.

Theorem 3. *Let χ be an arbitrary set and assume that B is the space of bounded functions over χ, $B(\chi)$. $T\colon B(\chi) \rightarrow B(\chi)$. 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.*

We now show that the coarse-grained learning algorithm governed by Equation (20) converges.

Proposition 1. *Suppose that the updating process of the coarse-grained learning algorithm follows Equation (20). The algorithm converges to the aggregated quality function* $Q(B^S, B^{S'}, B^M)$ *defined in Equation (10) if the following three conditions are met, and the initial learning rate* γ_0 *is small enough*:[2]

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, we can define a sequence of random operators $\tilde{\Gamma} = (\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_n, \ldots)$, with each operator takes the form:

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

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 $(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})$.

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

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

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

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

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.

We conclude that 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

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

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

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.
$$
\begin{aligned}
&\gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \\
&\times \left| \sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \|\Psi(B^{S'}, B^{S''}, B^{M'})\| \cdot \begin{pmatrix} Q^*(B^{S'}, B^{S''}, B^{M'}) - \\ Q(B^{S'}, B^{S''}, B^{M'}) \end{pmatrix} \right| \\
&\le \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \\
&\times \sum_{(B^{S'}, B^{S''}, B^{M'}) \in \omega(B^{S'})} \|\Psi(B^{S'}, B^{S''}, B^{M'})\| \begin{vmatrix} Q^*(B^{S'}, B^{S''}, B^{M'}) - \\ Q(B^{S'}, B^{S''}, B^{M'}) \end{vmatrix} \\
&\le \gamma_t \cdot \alpha \cdot \frac{1}{\varphi(B^{S'}) \cdot \|\Psi(B^S, B^{S'}, B^M)\|} \cdot \|\omega(B^{S'})\| \\
&\times \Psi'' \cdot \|Q^*(B^{S'}, B^{S''}, B^{M'}) - Q(B^{S'}, B^{S''}, B^{M'})\|_\infty
\end{aligned}
$$

$\| \cdot \|_\infty$ 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

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

and defining the function F_t as

$$F_t = \gamma_t \cdot Con$$

we obtain the inequality

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

Under the assumption that Con < 1 or $F_t|_{t=0} < 1$, Condition 2 of Theorem 4 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 $\gamma_0 \cdot \text{Con} < 1$.

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

$$G_t = \frac{t-1}{t}, \text{ so that}$$

$$\prod_{t=k}^{n} = \frac{k-1}{k} \cdot \frac{k}{k+1} \cdots \frac{n-1}{n} = \frac{k-1}{n}$$

$$\lim_{n \to \infty} \left(\prod_{t=k}^{n} G_t \right) = 0$$

Condition 3 is therefore satisfied.

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

The four conditions of Theorem 4 under proper assumptions are all met, and we conclude that the coarse-grained learning algorithm converges to the aggregated quality function defined in Equation (10). □

We now show that the coarse-grained learning algorithm governed by Equation (22) converges.

Proposition 2. *If the updating process of the coarse-grained learning algorithm follows the Equation (22), the algorithm converges.*

Proof. To show that the updating process described by Equation (22) finally converges, it is suffice to consider the update carried out for any triplet $(B^S, B^{S'}, B^M)$. We first re-write Equation (22) as

$$\begin{aligned} Q(B^S, B^{S'}, B^M) \leftarrow{} & (1-\gamma_t)\cdot Q(B^S, B^{S'}, B^M) \\ & + \gamma_t \cdot \left[(HQ)(B^S, B^{S'}, B^M) + w_t(B^S, B^{S'}, B^M)\right] \end{aligned}$$

where (HQ) is defined as

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

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

$$\begin{aligned} & w_t(B^S, B^{S'}, B^M) \\ & \quad = R(S,S'')|_{S''\in B^{S''}} + \alpha \\ & \quad \times \underset{(B^{S''},B^{S'''},B^{M'})\in\omega(B^{S''})}{\text{Avg}} Q(B^{S''}, B^{S'''}, B^{M'}) - (HQ)(B^S, B^{S'}, B^M) \end{aligned}$$

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

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

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

$$
\begin{aligned}
&|(HQ)(B^S, B^{S'}, B^M) - (H\bar{Q})(B^S, B^{S'}, B^M)| \\
&\leq \alpha \cdot \sum_{B^{S''}} \left[\begin{array}{c} Pr(B^{S''}|B^S, B^{S'}, B^M) \cdot \\ \max \left| \begin{array}{c} Q(B^{S''}, B^{S'''}, B^{M'}) - \\ \bar{Q}(B^{S''}, B^{S'''}, B^{M'}) \end{array} \right| \end{array} \right] \\
&\leq \alpha \cdot \max \left| Q(B^{S''}, B^{S'''}, B^{M'}) - \bar{Q}(B^{S''}, B^{S'''}, B^{M'}) \right| \\
&\leq \alpha \cdot \|Q - \bar{Q}\|_\infty
\end{aligned}
\tag{39}
$$

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 (39), we now conclude that

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

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 [7]), the updating process described by Equation (22) 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 (10). □

We notice that our proof of Proposition 2 does not rely on assumptions A1–A4 made in Section 3.1. This observation shows that our coarse-grained learning algorithm convergent 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*[3] *and the learning rate γ_m (as in Equation (35)) satisfies*:

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

In order to prove Proposition 4, we need to exploit one theoretical result from the stochastic approximation research [15,30], which 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*

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

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 4, we now prove Proposition 3.

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

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

Obviously, the updating process determined by the above equation is equivalent to the fine-grained updating rule given in Equations (35) and (36). Making this statement, we have implicitly utilized assumption A3, 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 (40) 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, we arrive at a more compact representation of the updating process as in Equation (41).

$$\underset{(S_{i+2}, M_{i+1}, (G_M)_{i+1})}{\text{Avg}} \left(\phi(S_{i+1}, S_{i+2}, M_{i+1}, (G_M)_{i+1}) \right) \tag{40}$$

$$\vec{b} \leftarrow \vec{b} + \gamma_t \sum_{i=0}^{\mathcal{N}-1} \left\{ Z_i \cdot \left(\begin{array}{c} R(S_i, S_{i+1}) + \alpha \cdot \tilde{\phi}(S_i, S_{i+1}, M_i, (G_M)_i)^T \cdot \vec{b} - \\ \phi(S_i, S_{i+1}, M_i, (G_M)_i)^T \cdot \vec{b} \end{array} \right) \right\} \tag{41}$$

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

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

We simplify the updating process given in Equation (41) as

$$\vec{b} \leftarrow \vec{b} + \gamma_t (A \cdot \vec{b} + D + w) \tag{42}$$

where w is a zero mean noise term. By utilizing Theorem 4, 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, we have

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

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 $\mathcal{M}$.

We also 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, we can obtain the following two equations, which is proved in [7].

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

and

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

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, we further represent matrix A as

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

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 (40)), we can establish a relation between $\tilde{\Phi}$ and Φ 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

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

Using this equation, we have

$$A = \Phi^T \mathcal{Q} \sum_{i=0}^{\infty} \Big[\lambda^i (\alpha P)^{i+1}\Big] \Phi - \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\lambda P)^i\right] \Phi + \phi^T \mathcal{Q} \alpha \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \Phi$$

$$\begin{aligned} =& \Phi^T \mathcal{Q} \left[(1-\lambda)\sum_{i=0}^{\infty} \Big(\lambda^i (\alpha P)^{i+1}\Big) - I\right] \Phi + \alpha \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \\ =& \Phi^T \mathcal{Q} (M - I) \Phi + \alpha \Phi^T \mathcal{Q} \left[\sum_{i=0}^{\infty} (\alpha\lambda P)^i P_E\right] \Phi \end{aligned} \tag{43}$$

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, we 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 $\Delta Q'$. We now consider the multiplication performed on the first term of Equation (43).

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

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 [7], we have

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

and

$$\|P^m \vec{J}\|_{\mathcal{Q}} \le \|\vec{J}\|_{\mathcal{Q}}, \forall \vec{J}, \quad m \ge 0$$

Subsequently,

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

With the above inequality, we get

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

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

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

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), 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, we prove that by properly choosing the discount rate α, the matrix A is negative definite for all vectors $\vec{b}$ within the region *Re*. Consequently, our fine-grained learning algorithm with fixed learning policy (i.e., the fixed policy learning strategy) is convergent. This concludes the proof of Proposition 4. □

We end this section by presenting the fourth theoretical result regarding our fine-grained learning algorithm.

Proposition 4. *The fine-grained learning algorithm, which adopts an updating process governed by Equation (38), converges, provided that Proposition 4 is valid, the step-weight δ is properly selected, and the learning rate γ_t of Equation (38) satisfies the following conditions*:

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

Proof. To simplify our discussion, we consider only the case where the output vector $\vec{b}$ to be updated by Equation (35) is within a bounded region around $\vec{0}$. Actually, due to the trapping function trap(•), the vector $\vec{b}$, which is produced by Equation (35) and is outside the bounded region, is re-mapped into the region. Thus, we can re-write Equation (38) 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 [7]), if we can show that $H(\bullet)$ 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. We 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 (42). 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 predefined region (due to the trapping function), we use *Re* to denote the set of all possible vectors $\vec{b}$ that can be processed by the algorithm. We further define $E_A(\vec{b}_1, \vec{b}_2)$ 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:

$$\|H(\vec{b}_1) - H(\vec{b}_2)\| < \|\vec{b}_1 - \vec{b}_2\| \tag{44}$$

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

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

We can re-write inequality (44) as

$$\delta \cdot \|{}^2\vec{b}_1 - {}^2\vec{b}_2\| < \|\vec{b}_1 - \vec{b}_2\| \tag{45}$$

Consequently, to show that $H(\bullet)$ is contractive is to show that inequality (45) is valid for all $\vec{b}_1, \vec{b}_2 \in Re$. Derived from the error bounds on solutions of linear systems [25], we have

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

where X and Y are two positive numbers. Since both functions E_A and E_D are continuous and bounded for all $\vec{b}_1, \vec{b}_2 \in Re$, we can find a step-weight δ such that inequality (45) is valid. Thus, we conclude that by choosing a proper δ, the fine-grained learning algorithm under our restricted policy iterations, is convergent. □

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 (38) 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, we present the experimental studies of the learning algorithms in Section 5.

5. Experimentation results: A multiagent pathology lab system

To demonstrate the effectiveness of our learning algorithms, we consider a multiagent pathology lab system. A pathology lab is considered as 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 10, and a simulated multiagent system is developed to show the modeling technique and experiment with the coordination algorithms.

In Figure 10, 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 (*Agent*1 and *Agent*2) are deployed. *Agent*1 manages two centrifugal and one freezing devices. The rest of devices (Figure 10) are managed

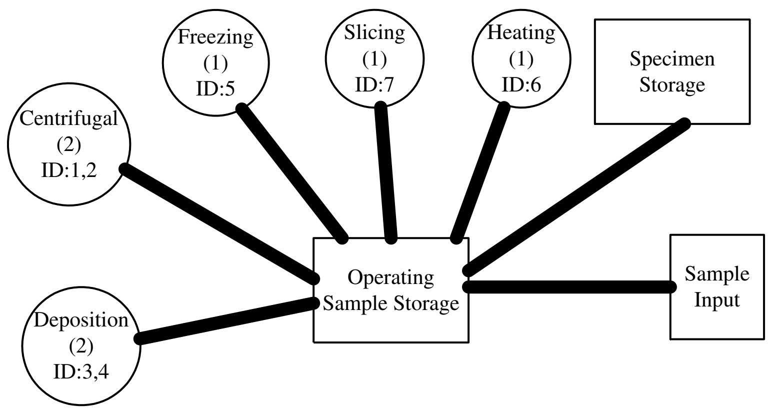

Figure 10. The structure of a simplified pathology lab.

by *Agent*2. 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

$$\text{Similar}_{\overline{M}}(M) = \mu_{\tilde{A_c}}(a_c) \bigwedge \mu_{\tilde{E_q}}(r) \bigwedge \mu_{\tilde{O}}(o) > 0$$

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* [43]. 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; (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 five categories. Table 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 2 lists all the meta-methods of a task that produces a specimen from a sample o of type $A.1$. In Table 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 $\tilde{A_c}$.

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 2 are modeled as a *directed graph* (Figure 11), which can be viewed as a workflow. 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.

Table 1. All applicable actions, their action category and processing time.

Action Category	Centrifugal operation	Deposition operation	Freezing operation	Heating operation	Slicing operation
Processing Time and 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 2. The set of meta-methods of task T for large sizes sample of type A.

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}	$\{o\}$

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 12.

We use Equation (1) and the graph similarity measure (Section 3.4) to determine the degree to which extent a method is relevant to certain hard and soft relations. The degree is interpreted as the *possibility* that the method may be affected by the corresponding relations, and takes its value between 0 and 1. The *possibility* belongs to the agents' subjective knowledge. The actual impact is estimated through our learning algorithms.

Each time our simulated system starts, a total number of five 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 (46) is used to evaluate the reward incurred when the task concludes.

$$Reward_T = \begin{cases} rFac - pFac & when\ task\ T\ is\ fulfilled \\ failPenalty & when\ task\ T\ fails \end{cases} \tag{46}$$

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 (47):

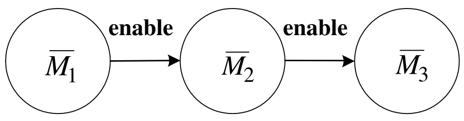

Figure 11. The *hard relations* among the meta-methods of task T (Table 2).

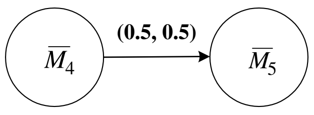

Figure 12. The *soft relation* between centrifugal and deposition meta-methods.

$$
\begin{aligned}
rFac &= Str \cdot \left(e^{\frac{t_b - t}{t_b} \cdot rDis} \right) \\
pFac &= Str \cdot \left(e^{\frac{t_b - t}{t_b} \cdot pDis} - e^{2 \cdot \frac{t - t_b}{t_b}} \right)
\end{aligned}
\tag{47}
$$

In (47), 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). The discount rate $\alpha(t)$ is defined as $\alpha(t) = e^{-\alpha^c \cdot t}$, where α^c is a given discount constant. Table 3 summarizes the actual values for these constants used in the experiment. As shown in Table 3, each task failure results in a heavy penalty, highlighting the importance of managing hard system constraints.

To investigate the effectiveness of our learning algorithms, we carried out several experiments. 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 3.3) in our 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. They are obtained by averaging 500 independent evaluations of the learning algorithm.

Our experiment shows that the learning result converges after 5000 times of learning (Table 4). 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. We notice that this performance result can be further improved if the coarse-grained learning algorithm exploits extra fluents, as our 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 5.

Table 3. Constants used in the evaluation of rewards and the system performance.

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

Table 4. The performance of the coarse-grained learning algorithm by one learning agent.

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

Compared with data in Tables 4 and 5 indicate 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 our third experiment, we try to find out whether the system performance can be 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 expected 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 6.

Table 6 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 Tables 4–6 are compared in Figure 13.

To investigate the scalability of the algorithm in terms of the number of learning agents, we have carried out similar experiments for 3–8 agents. A total of three 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 7, and the changes of performance with respect to the number of learning agents is shown in Figure 14.

Table 7 and Figure 14 show that by employing more learning agents, the system performance is further improved. Nevertheless, no extra performance improvement can be observed after five learning agents are used. This is not surprising as within

Table 5. The performance of the coarse-grained learning algorithm for the situation of two learning agents.

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

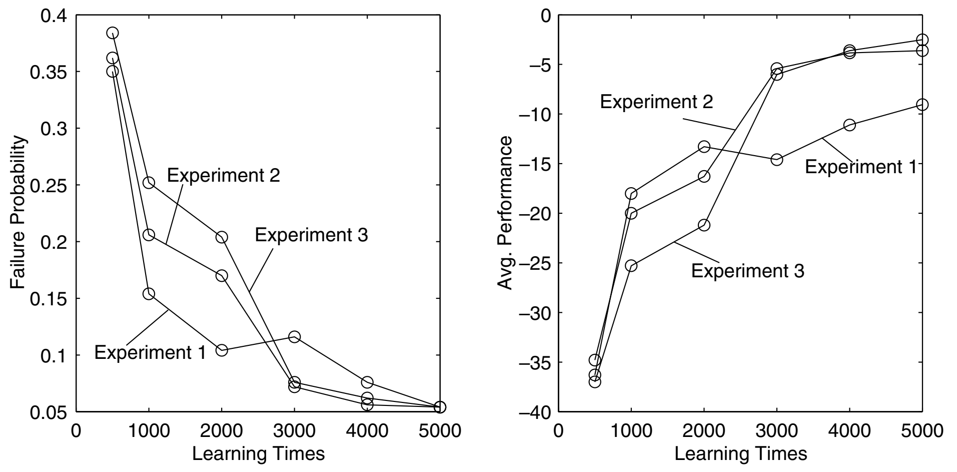

Figure 13. Performance of the coarse-grained learning algorithm.

Table 6. The performance of the coarse-grained learning algorithm with an extended fluent set and two learning agents.

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

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

Besides showing that our learning algorithms are effective, the algorithms also appear to be good choices for task-oriented domains. We specifically compared our learning results with a genetic algorithms-based learning scheme similar to the one proposed in [8]. Instead of using reinforcement learning methods, this approach uses the genetic algorithms to adjust the fuzzy rules. Within the fuzzy

Table 7. The performance of the coarse-grained learning algorithm with respect to the number of learning agents.

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

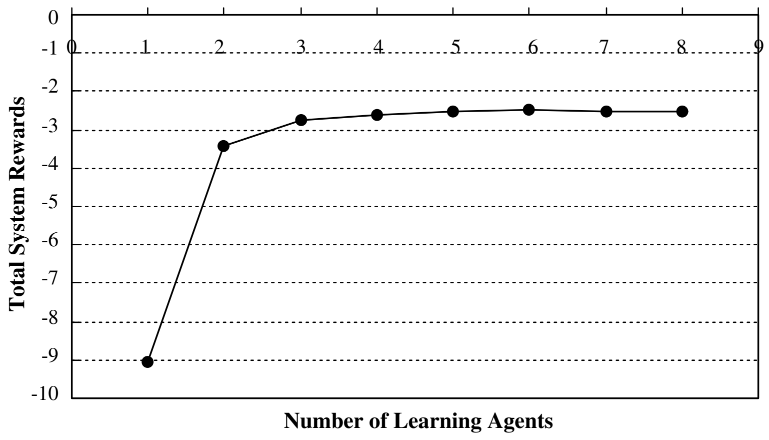

Figure 14. The changes of performance with respect to the number of learning agents.

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 our implementation of the scheme, 20 chromosomes form a generation. In order to carry out one generation iteration, we must evaluate each of the 20 chromosomes. For any chromosome to be evaluated, we first adjust each fuzzy rule output according to the chromosome. The derived fuzzy rule base is consequently employed by each agent (two agents are deployed in our experiment) of our 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, we use an average of 50 independently observed system performances as the evaluated fitness. Given this brief description of the *GA-based scheme*, Figure 15 shows the average system performance achieved with respect to 50 consecutive generation iterations.

As shown in Figure 15, 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 our coarse-grained learning algorithm.

Compared with our experiments described previously, at least two major advantages of our 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, our learning

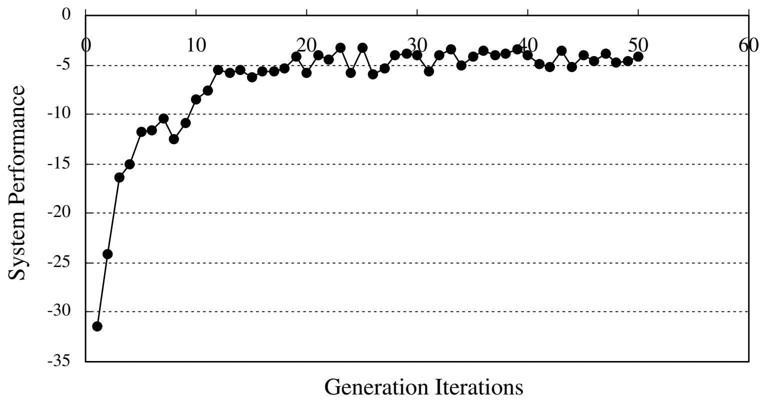

Figure 15. The average system performance achieved for 50 consecutive generation iterations.

algorithms can offer certain convergence guarantees as the theoretical analysis (Section 4) has already shown.

2. The computational complexity of the genetic algorithm tends to be significantly higher than that of our 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, our coarse-grained learning algorithm requires only 4000 independent simulations. These results suggest that our learning algorithms can be five times faster than the GA-based scheme.

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

We now present the last two experiments which are developed for the fine-grained learning algorithm. The 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 (Figure 12). The learning results obtained from the second experiment are utilized to facilitate the fine-grained learning process. Figure 16 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 (Section 3.4).

Figure 16 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 our simulated system contributes to a faster convergence rate and a higher system performance. Table 8 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, we are convinced that the fine-grained learning algorithm is effective in managing soft system constraints.

Figure 17 shows the performance of our fine-grained learning algorithm under the restricted policy iterations (Section 3.4). As implemented in this experiment, during

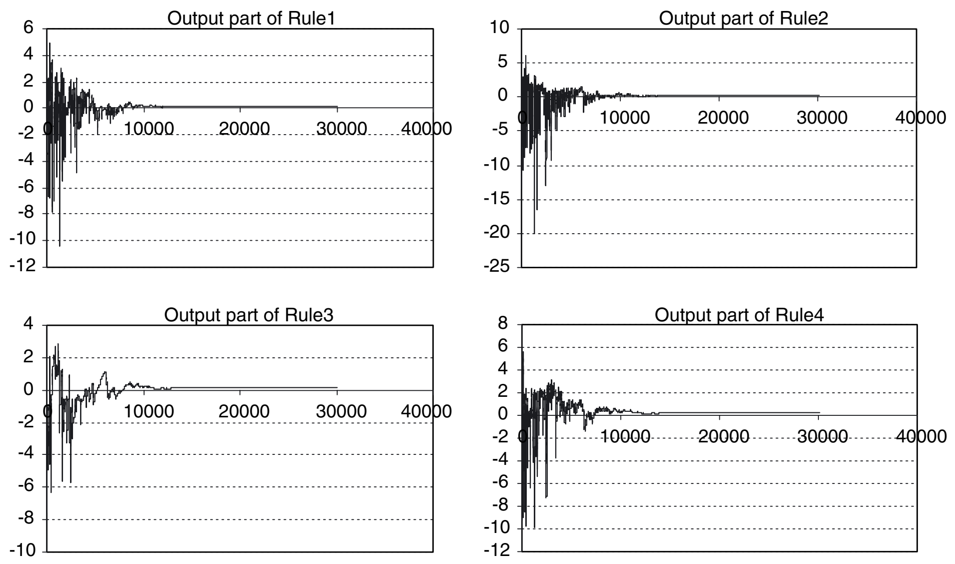

Figure 16. The changing process of the four fuzzy rule outputs. The fine-grained learning algorithm takes the fixed policy learning strategy. The horizontal axis denotes the times of learning.

each learning phase, the updating procedure determined by Equation (35) is carried out only once. Reader may notice that, as indicated in Section 4, this treatment would not prevent the algorithm from converging, assuming that $H(\bullet)$ is a contractive mapping. The specific configuration is as follows: v 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.

As compared with Figures 16 and 17 show 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 our restricted policy iteration is more time efficient than the fixed policy learning strategy. Nevertheless, contrary to our 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

Table 8. The performance of the fine-grained learning algorithm.

λ	Failure probability	Average performance improvement
0.3	0.054	1.58
0.5	0.0538	2.31
0.9	0.054	1.73

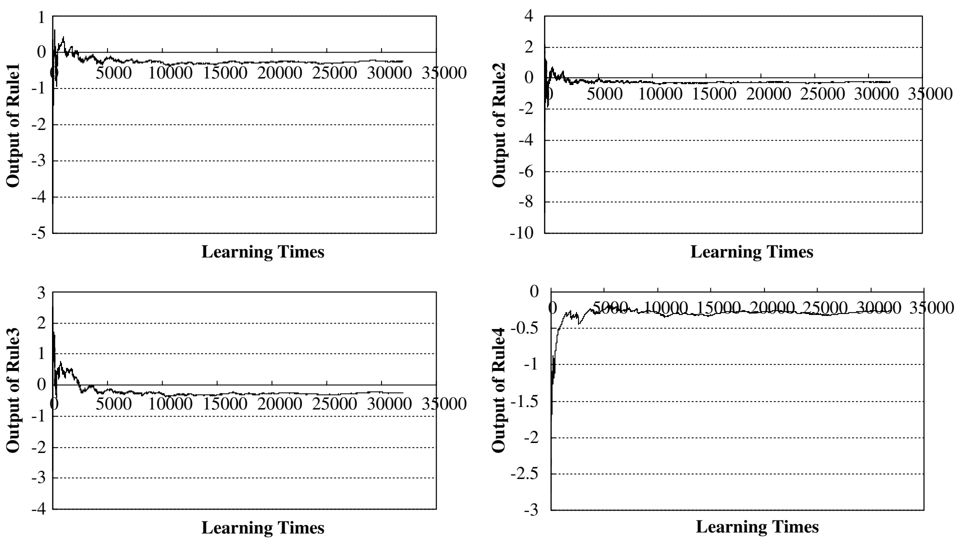

Figure 17. The changing process of the four fuzzy rule outputs. The fine-grained learning algorithm takes the *restricted policy iteration* learning strategy.

outputs learned through Equation (35), should take the blame. However, for some applications, the step-weight δ is necessary to ensure that the algorithm is convergent. With these results, we believe that the two learning strategies of our fine-grained learning algorithm are equally important, and they will find their practical use in various application domains.

6. Related work

The agent coordination issues have been originally addressed in the Distributed Artificial Intelligence (DAI) research [9]. Coordination among agents is usually achieved through distributed planning and scheduling techniques [19, 21, 32]. Recently, agent coordination via reinforcement learning has received much research interests [13, 27, 36, 38, 41]. 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 [7]. 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 [16, 34]. For example, the Q-learning algorithm has been exploited to coordinate multiple agents controlling elevators [16, 40]. The learning results are stored in a neural network, and the Q-learning algorithm constantly adjusts network's weights. Different from this approach, our work 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, our approach is capable of handling a variety of system fluents, which take either numerical or symbolic values. The adoption of aggregation techniques enable our 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 [1, 2, 5, 33]. Similar to ours, much of these works is centered around the issue of job and resource allocations among multiple agents [5, 41]. 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 [41]. On the other hand, in the work of Abul et al. the effects of other agents' behaviors can only be acquired through self-observations [1]. 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 [5], 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 [5] 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 interdependencies among agents) in learning algorithms was noticed in the literature, such as [39]. 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.

In discussing the related work, it often helps if some statements can be made on a possible *equivalence* between the related work and ours. We realize that 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 the rest of this section to present one perspective on the "equivalent approaches".

6.1. 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* [22]. Within our context, 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, we discuss two related work reported in the literature that can be seen as equivalent to ours. For the detailed description of these two works, reader is referred to [23, 39].

Vengerov et al. proposed an adaptive coordination approach to distributed dynamic load balancing among fuzzy reinforcement learning agents [39]. 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, our 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 [39] 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 [39] serves as an application example of our learning approaches.

Notice that the concept of the method relations was not clearly identified in [39]. This shortage was amended in [23]. Guestrin et al. considered the problem of context specific multiagent coordination and planning with Factored MDPs [23]. 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 [23]. In this aspect, we cannot compare their approaches with our learning algorithms.

7. Conclusion

In this paper, we explored reinforcement learning methods for coordinating multiple agents in a task-oriented environment. We presented the FSTS to formally model agent coordination problems. We showed that 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 agents coordination, where the 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 our 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, our algorithms appear to be good learning approaches for task-oriented environments.

The main contributions of this paper are summarized as follows. First, our approach applies to *general* agent coordination problems in task-oriented environments and is not limited to any specific application scenario. Second, we deal with the overall coordination problem through the combination of two learning algorithms, which operate at different levels of abstraction. The algorithms were formally proved to converge and experimentally shown to be effective. Finally, a systematic approach to modeling and utilizing system constraint information is proposed; and constraint information is explicitly incorporated into the reinforcement learning process. This approach towards system constraints underpins the two algorithms, and attributes to the effectiveness of the algorithms.

This paper also opens the doors for future investigation. We take note that the agents' decision policies may change wildly with slight changes of task-oriented environments, and thus another approximation function for estimating the changing process of agents' interdependencies may be necessary to be built into our FSTS model and then utilized explicitly by the learning algorithms. Furthermore, we seek to define formally the *equivalence* notion so that certain learning approaches reported in the literature can be shown actually *equivalent* to our methods.

Acknowledgments

The authors would like to thank the anonymous referees for their insightful comments that helped significantly improved this paper.

Notes

1. In this paper, we do not include both hard and soft relations in the same graph.

2. What exactly is meant by "small enough" will become clear in the proof of this proposition.
3. The conditions satisfied by the discount rate α will be clarified during the proof of this proposition.

References

1. O. Abul, F. Polat, and R. Alhajj, "Multiagent reinforcement learning using function approximation" *IEEE Trans. Syst., Man, Cyber*. vol. 30, no. 4, pp. 485–497, 2000.
2. S. Arai, K. Sycara, and T. R. Payne, "Multi-agent reinforcement learning for scheduling multiple-goals," in *Proceedings of the fourth International Conference on Multi-Agent Systems*, 2000.
3. R. Bellman, *Dynamic Programming*, Princeton University Press: Englewood Cliffs, NJ, 1957a.
4. R. E. Bellman, *Dynamic Programming*, Princeton University Press: Englewood Cliffs, NJ, 1957b.
5. 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.
6. D. P. Bertsekas, *Dynamic Programming*, Prentice-Hall: Englewood Cliffs, NJ, 1987.
7. D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996.
8. A. Bonarini and V. Trianni, "Learning fuzzy classifier systems for multi-agent coordination," *Int. J. of Inform. Sci*. pp. 215–239, 2001.
9. A. H. Bound and L. Gasser (eds.), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann: Los Atlas, CA, 1988.
10. C. Boutilier, T. Dean, and S. Hanks, "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage," *J. Artif. Intelligence Res*. Vol. 11, pp. 1–94, 1999.
11. C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artif. Intell*. Vol. 121, pp. 49–107, 2000.
12. H. Bunke and X. Jiang, "Graph matching and similarity," in H. N. Teodorescu, D. Mlynek, A. Kandel, and H. J. Zimmermann (eds.), *Intelligent Systems and Interfaces*, Kluwer Academic Publishers: Dordrecht, 2000.
13. 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.
14. G. Chen, Z. Yang, H. He, and K. M. Goh, "A fuzzy logic based multiagent coordination framewrok," in *Proceedings of the International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, 2003, Vienna, Austria.
15. H. Chen, *Stochastic approximation and its Applications*, Kluwer Academic Publishers: Dordrecht, 2002.
16. R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine Learning*, vol. 33, pp. 235–262, 1998.
17. R. Dearden and C. Boutilier, "Abstraction and approximate decision-theoretic planning," *Artif. Intell*., vol. 89, pp. 219–283, 1997.
18. K. S. Decker "Environment centered analysis and design of coordination mechanisms," Ph.D. thesis, University of Massachusetts Amherst, 1995.
19. K. S. Decker and V. R. Lesser, "Generalizing the partial global planning algorithm," *Int. J. Intell. Cooperative Inform. Syst*., pp. 319–346, 1992.
20. E. H. Durfee, (ed.), "*Coordination of Distributed Problem Solvers*," Kluwer Academic Publishers: Dordrecht, 1988.
21. E. H. Durfee and V. R. Lesser, "Negotiation task decomposition and allocation using partial global planning," in *Distributed Artificial Intelligence*, vol. 2, 1989, pp. 229–243.
22. R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in Markov decision processes," *Artif. Intell*. vol. 147, pp. 163–223, 2003.
23. 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, 2002.

24. M. Heger, "Consideration of risk in reinforcement learning," in *Proceedings of the 11th International Conference on Machine Learning*, 1994. pp. 105–111.
25. R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press: Cambridge, MA, 1985.
26. L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A survey," *J. Artif. Intell. Res.* vol. 4, pp. 237–285, 1996.
27. S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Eighteenth National Conference on Artificial Intelligence*, 2002. Edmonton, Alberta, Canada, pp. 326–331.
28. T. W. Malone and K. Crowston, "What is coordination theory and how can it help design cooperative work systems?," in *Proceedings of the 1990 ACM conference on Computer-Supported Cooperative Work*, 1990. pp. 357–370.
29. M. L. Puterman, "Markov decision processes," in D. P. Heyman and M. J. Sobel (eds.), *Handbook in Operations Research and Management Science*, vol. 2, 1990. Stochastic Models. North-Holand, Chapt. 8, pp. 331–434.
30. H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.* vol. 22, pp. 400–407, 1951.
31. J. Rust, "Numerical dynamic programming in economics," in H. M. Amman, D. A. Kendrick, and J. Rust (eds.), *Handbook of Computational Economics*, vol. 1, 1996. Amsterdam, Elsevier, Amsterdam; The Netherlands: Chapt. 14.
32. A. Sathi, and M. S. Fox, "Constraint-directed negotiation of resource reallocations," in *Distributed Artificial Intelligence*, vol. 2, 1989. pp. 163–193.
33. S. Sen and M. Sekaran, "Individual learning of coordination knowledge," *J. Exp. Theor. Artif. Intell.* vol. 10, pp. 333–356, 1998.
34. P. Stone, "Layered learning in multi-agent systems," Ph.D. thesis, School of Computer Science, Carnegie Mellon University, 1998.
35. M. Sugeno, "An introductory survey of fuzzy control," *Inform. Sci.* vol. 36, pp. 59–83, 1985.
36. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.
37. C. Szepesv*á*ri and M. L. Littman, "A unified analysis of value-function-based reinforcement learning algorithms," *Neural Comput.* vol. 11, no. 8, pp. 2017–2060, 1996.
38. M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative learning," in M. N. Huhns and M. P. Singh (eds.), *Readings in Agents*, San Francisco, CA, USA: Morgan Kaufmann, 1997. pp. 487–494.
39. 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.
40. C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.
41. 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.
42. D. J. White, *Markov Decision Processes*, John Wiley & Sons: New York, 1993.
43. L. A. Zadeh, in L. A. Zadeh, R. R. Yage and R. R. Yager and R. M. Ton (eds.), *Fuzzy Sets and Applications: Selected Papers*, John Wiley & Sons, New York, 1987.