Learning Classifier Systems Meet Multiagent Environments

Keiki Takadama[1], Takao Terano[2], and Katsunori Shimohara[3]

[1] ATR International, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288 Japan
Tel: +81-774-95-1007, Fax: +81-774-95-2647
[2] Univ. of Tsukuba, 3-29-1, Otsuka, Bunkyo-ku, Tokyo 112-0012 Japan
Tel: +81-3-3942-6855, Fax: +81-3-3942-6829
[3] ATR International, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288 Japan
Tel: +81-774-95-1070, Fax: +81-774-95-2647
{keiki,katsu}isd.atr.co.jp, terano@gssm.otsuka.tsukuba.ac.jp

Abstract. An Organizational-learning oriented Classifier System(OCS) is an extension of Learning Classifier Systems (LCSs) to multiagent environments, introducing the concepts of organizational learning (OL) in organization and management science. Unlike conventional research on LCSs which mainly focuses on single agent environments, OCS has an architecture for addressing multiagent environments. Through intensive experiments on a complex scalable domain, the following implications have been revealed: (1) OCS finds good solutions at small computational costs in comparison with conventional LCSs, namely the Michigan and Pittsburgh approaches; (2) the learning mechanisms at the organizational level contribute to improving the performance in multiagent environments; (3) an estimation of environmental situations and utilization of records of past situations/actions must be implemented at the organizational level to cope with non-Markov properties in multiagent environments.

Keywords: learning classifier system, multiagent system, non-Markov environment, organizational learning

1 Introduction

Currently, *non-Markov* environments are studied on in the context of Learning Classifier Systems (LCSs) [Goldberg 89], and the capability of LCSs in such environments is investigated in the framework of *reinforcement learning* [Sutton 98]. In particular, these researches mostly address the POMDP (partially observable Markov decision process) environments where an agent cannot distinguish different situations due to a lack of global environmental information. This non-Markov property is related to a location in single agent environments. As a major approach towards such environments, estimation of environmental situations, which is a model-based approach, showed its effectiveness in the POMDP environments [Chirsman 92,MacCallum 93], while utilization of records of past

P.L. Lanzi, W. Stolzmann, and S.W. Wilson (Eds.): IWLCS 2000, LNAI 1996, pp. 192–210, 2001.
© Springer-Verlag Berlin Heidelberg 2001

situations/actions contributed to coping with such environments by adding temporary memory [Cliff 95,Lanzi 98,Lanzi 99].

However, the property in the POMDP environments is one of non-Markov properties and another property are still embedded in multiagent environments [Weiss 99]. Another property is peculiar in multiagent environments and is related to a change of an agent's internal state. Specifically, this property makes it difficult for an agent to recognize an environmental change caused by the change of another agent's internal state, due to a lack of the other agents' information. From this characteristic, the effective actions of the agent vary due to such kinds of changes even if the agent stays in the same location. Here, defining this environment as the NOMDP (non observable Markov decision process) environment, the non-Markov properties in multiagent environments include both properties in the POMDP and NOMDP environments. Thus, addressing multiagent environments is more complex than addressing single agent environments.

Due to these difficulties, there is no standard and promising method to address the above non-Markov properties in multiagent environments. Therefore, as one of the methods, this paper explores elements needed in LCSs to cope with multiagent environments using our Organizational-learning oriented Classifier System (OCS) [Takadama 99] which is an extension of LCS to multiagent environments. Concretely, we investigate such elements by analyzing results that compare the performance of OCS in multiagent environments with those of the conventional LCSs, namely the *Michigan* [Holland 78] and *Pittsburgh* [Smith 83] approaches.

This paper is organized as follows. Section 2 starts by describing organizational learning (OL) which is introduced in our LCS. Section 3 explains the architecture of OCS. A multiagent example is given in Sect. 4. Sections 5 shows simulations of a comparison between OCS and conventional LCSs. Section 6 discusses elements for coping with multiagent environments. Finally, our conclusions are made in Sect. 7.

2 Organizational Learning

2.1 Four Loop Learning in Organizational Learning

Organizational learning (OL) [Argyris 78,Cohen 95] has been studied in the context of organization and management science and is roughly characterized as organizational activities that improve organizational performance which cannot be achieved at an individual level. In particular, OL consists of the following four kinds of learning [Kim 93,Argyris 78].

- **Individual single-loop learning** improves the performance within an individual norm.
- **Individual double-loop learning** improves the performance through the change of an individual norm itself.
- **Organizational single-loop learning** improves the performance within an organizational norm.

– **Organizational double-loop learning** improves the performance through the change of an organizational norm itself.

2.2 Reinterpretation of Four Loop Learning

The categorization in the previous section stipulates that (1) there are individual and organizational levels in learning, and (2) each learning can be classified as a single or a double type. However, the term *norm*[1] in the above learning has not been defined clearly from a computational viewpoint. This allows us to implement such learning in many ways from a computational viewpoint, and this way of implementation may lead to different results. Since such ambiguity is not appropriate to discuss focused issues consistently, this paper starts by assuming the norms as follows.

– **Individual norm** is implemented by individual knowledge.
– **Organizational norm** is implemented by organizational knowledge.

Next, these two types of knowledge are assumed as follows.

– **Individual knowledge** is implemented by a rule set.
– **Organizational knowledge** is implemented by a set of individual knowledge (rule set).

These assumptions may seem to define only parts of organizational learning, but they come from the consideration that individual (or organizational) norms are kinds of behavior established in an individual (or an organization) and these behaviors are derived from individual (or organizational) knowledge. By clarifying such norms through the above assumptions, we can potentially find new results which cannot be derived from the conventional definitions. Based on this claim, this paper defines *computational organizational learning* as "learning that includes four kinds of computationally interpreted loop learning".

3 Organizational-Learning Oriented Classifier System

An Organizational-learning oriented Classifier System (OCS) [Takadama 99] has GBML (Genetics–Based Machine Learning) architecture and includes multiple Learning Classifier Systems (LCSs) extended to introduce the concepts of organizational learning.

3.1 Agents

In OCS, agents are implemented by their own LCSs, which are extended to introduce four kinds of learning mechanisms in OL. In order to solve problems that cannot be solved at an individual level, agents cooperate with other agents

[1] For instance, a routine and a weltanschauung are examples of an individual norm and an organizational norm, respectively.

by specializing their own roles. This indicates that OCS is based on problem solving through role specialization. Since these roles are implemented by *rule sets* in agents' own LCSs, the *aim* of the agents is defined as acquiring appropriate *rule sets* through interaction with other agents. Specifically, the learning mechanisms make these rule sets appropriate by varying *if-then* rules and the strength[2] values of these rules.

3.2 Architecture

As shown in Fig. 1, OCS is composed of many agents, and each agent has the same architecture which includes the following components.

< **Problem Solver** >
- **Detector** and **Effector** translate a part of an environment state into an internal state of an agent and derive actions based on this internal state [Russell 95], respectively.

< **Memory** >
- **Organizational knowledge memory** stores a set comprising each agent's rule set as organizational knowledge. In OCS, this knowledge is shared by all agents and represents knowledge on role specializations.

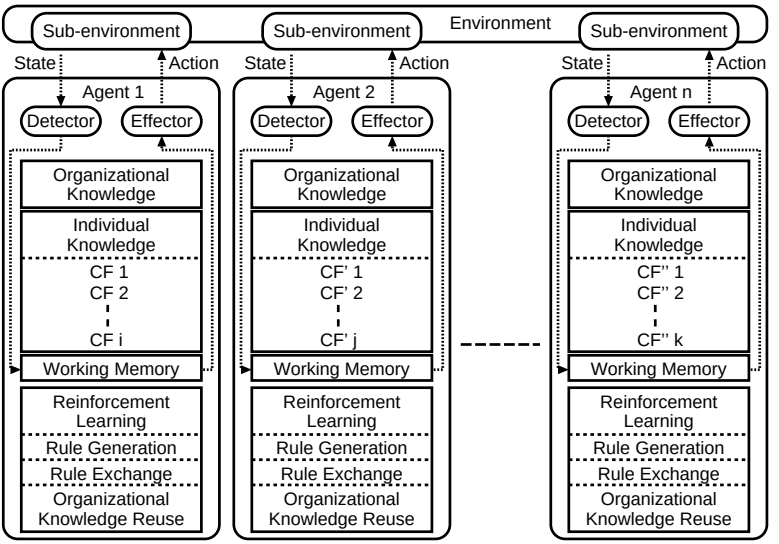

Fig. 1. OCS architecture

[2] The term strength in this paper is defined as the worth or weight of rules.

- **Individual knowledge memory** stores a rule set (a set of classifiers (CFs)) as individual knowledge. In OCS, agents independently store different CFs that are composed of *if-then* rules with a strength factor. In particular, a fixed number (`FIRST_CF`) of rules in each agent is generated at random in advance, and the strength values of all rules are set to the same initial value. In this case, one primitive action is included in the *then* part.
- **Working memory** stores the recognition results of sub-environmental states and also stores the internal states of actions of fired rules.

< **Mechanisms** >

- **Reinforcement learning mechanism**, **rule generation mechanism**, **rule exchange mechanism**, and **organizational knowledge reuse mechanism** are computationally interpreted from the four kinds of learning in OL (Details are described later). These mechanisms are not improved by specific or elaborate techniques but by simple and ordinary ones.

3.3 Learning in OCS

(1) Reinforcement Learning Mechanism. In OCS, the reinforcement learning (RL) mechanism enables agents to acquire their own appropriate actions that are required to solve given problems. In particular, RL supports the learning of the appropriate order of fired rules by changing the strength values of the rules. Since this mechanism improves problem solving efficiency at an individual level not by creating/deleting rules but by utilizing them while changing the order of the fired rules, it works as a kind of "individual single-loop learning," which is interpreted to improve performance within individual rules in computational organizational learning. To implement this mechanism, OCS employs a *profit sharing* method [Grefenstette 88] that reinforces the sequence of all rules when agents obtain some rewards.[3] The concrete mechanism of RL in OCS follows as shown in Fig. 4–1.

(2) Rule Generation Mechanism. The rule generation mechanism in OCS creates new rules when none of the stored rules match the current environmental state. This mechanism adapts to the current environment, and works as shown in Fig. 4–2. Concretely, the condition (if) part of a rule is created to reflect the current situation, the action (then) part is determined at random, and the strength value of the rule is set to the initial value. When the number of rules is `MAX_CF` (maximum number of rules), in particular, the rule with the lowest strength is removed and a new rule is generated. Since this mechanism improves the problem solving range at an individual level by creating/deleting rules, it works as a kind of "individual double-loop learning," which is interpreted to improve performance through the change of individual rules themselves in computational organizational learning.

[3] The detailed credit assignment in OCS was proposed in [Takadama 98].

In addition to the above basic mechanism, the strength value of the fired rule (*e.g.*, the No. i rule) temporarily decreases as $ST(i) = ST(i) - FN(i)$, where $ST(i)$ indicates the strength of the No. i rule and $FN(i)$ indicates the fired number of the No. i rule. In particular, $FN(i)$ is counted when the No. i rule is fired and is reset to 0 when the situation changes. With this mechanism, the strength value of fired rules decreases as long as the situation does not change as in deadlocked situations where the same rules are selected repeatedly. Then, these rules become candidates that capable of being replaced by new rules, while the strength value of these rules is recovered when the situation changes.

(3) Rule Exchange Mechanism. In OCS, agents exchange rules with other agents at particular time intervals (`GA_STEP`)[4] in order to solve given problems that cannot be solved at an individual level as shown in Fig. 4–3. Since this mechanism improves the problem solving efficiency at the organizational level not by creating/deleting a set comprising each agent's rule set but by utilizing it among the agents, this mechanism works as a kind of "organizational single-loop learning," which is interpreted to improve performance within a set comprising each agent's individual rules in computational organizational learning.

In this mechanism, a particular number ((the number of rules) $\times$ `GENERATION_GAP`[5]) of rules with low strength values is replaced by rules with high strength values between two arbitrary agents. For example, when agents X and Y are selected as shown in Fig. 2, the CFs in each agent are sorted by order of their strength values (upper CFs have high strength values). After this sorting, $CF_{j-2} \sim CF_j$ and $CF'_{k-2} \sim CF'_k$ in this case are replaced by $CF'_1 \sim CF'_3$ and $CF_1 \sim CF_3$, respectively. However, rules that have higher strength values than a particular value (`BORDER_ST`) are not replaced to avoid unnecessary rule exchanges. The strength values of replaced rules are reset to their initial values, because effective rules in some agents are not always effective for other agents in multiagent environments.

(4) Organizational Knowledge Reuse Mechanism. Finally, agents in OCS store a set comprising each agent's rule set (individual knowledge) as knowledge on the role specialization when they most effectively solve given problems.[6] After storing this knowledge, agents reuse it for future problem solving, instead of using the initial individual knowledge generated at random. This mechanism enables agents to use the best knowledge acquired previously. We call the organizational knowledge as knowledge on role specialization, because organizational knowledge is a set comprising each agent's rule set and each rule set works as one of the roles to solve given problems at the organizational level. The concrete mechanism of organizational knowledge reuse in OCS follows as shown in

[4] This step is defined in Sect. 4.2.

[5] The ratio of removed rules.

[6] Since the efficiency depends upon the problem, it is generally difficult to define the efficiency. However, as one of the methods, agents can be made to solve a given problem most effectively by measuring a "good solution" or a "small cost".

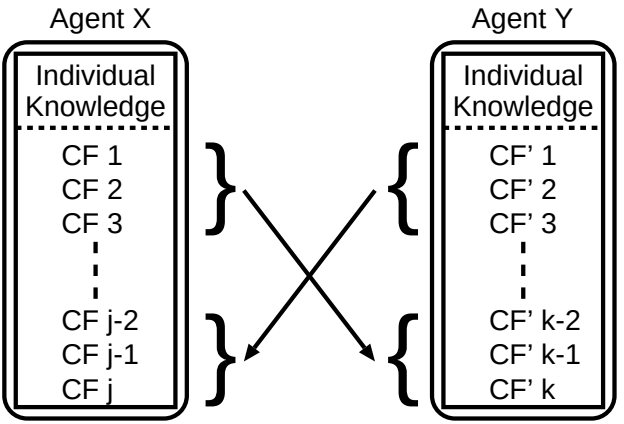

Fig. 2. Rule exchange

Fig. 4–4. Considering the situation in which n agents address a given problem, organizational knowledge (*i.e.*, a set comprising each agent's rule set) can be represented by {RS (1), RS (2), $\cdots$, RS (n)} as shown in Fig. 3, where RS(x) is the rule set for the x-th agent. Since this mechanism improves the problem solving range at an organizational level by creating/deleting organizational knowledge, it works as a kind of "organizational double-loop learning," which is interpreted to improve performance through the change of a set comprising each agent's individual rules itself in computational organizational learning.

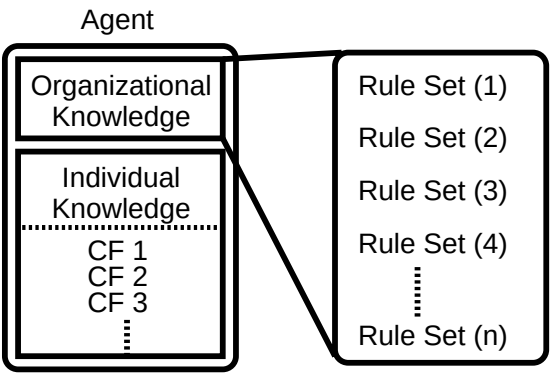

Fig. 3. Organizational knowledge reuse

```
procedure reinforcement learning
  begin
    if solution is converged then
      for all agents do
        fired rules are reinforced;
  end
```

Figure 4–1

```
procedure rule generation
  begin
    for all agents do
      if no matched rules then
        begin
          if number of rules = MAX_CF then
            the rule with the lowest strength value is deleted;
          a new rule is created;
          a strength value of the new rule is set to an initial one;
        end
  end
```

Figure 4–2

```
procedure rule exchange
  begin
    if mod (step, GA_STEP)=0 then
      for all pairs of agents do
        for (number of rules)×GENERATION_GAP rules do
          if the lowest strength value of rule ≤ BORDER_ST then
            begin
              a rule with a low strength value is replaced by
                a rule with a high strength value between two agents;
              a strength value of the replaced rule is reset to its
                initial value;
            end
  end
```

Figure 4–3

```
procedure organizational knowledge reuse
  begin
    if iteration=0 then
      stored organizational knowledge is utilized;
    else if solution is the best then
      begin
        if organizational knowledge is stored then
          stored organizational knowledge is deleted;
        current organizational knowledge is stored;
      end
  end
```

Figure 4–4

Fig. 4. Algorithms of four learning mechanisms

Furthermore, other characteristics of organizational knowledge are summarized as follows:

- Each agent at the current stage of OCS does not store the entire individual rule sets of all of the other agents independently, but shares the rule sets of all agents with other agents.
- Organizational knowledge is different from ordinary effective knowledge in a single LCS, because the former knowledge represents role specialization and is used in the unit of *multiple agents*, while the latter knowledge is utilized in the unit of *one agent*. Note that organizational knowledge is composed of a lot of redundant rules, but each agent does not reuse this knowledge by itself. Each agent reuses a part of the organizational knowledge, which means the knowledge that the same or other agent acquired in previous problem solving.
- Agents cannot use both individual and organizational knowledge at the same time, because the former knowledge is modified by each agent *during* problem solving while the latter knowledge is stored or reused by all agent *after* or *before* problem solving.

3.4 Relationships among the Four Learning Mechanisms

The total algorithm of OCS follows the procedure shown in Fig. 5. Briefly, organizational knowledge is reused if it is stored before agents solve a problem, and then, both the rule generation and rule exchange mechanisms are executed until the solution converge. After the convergence of the solution, both the reinforcement learning and organizational knowledge reuse mechanisms are executed, and agents continue to address the same problem until the iteration arrives at the maximum number of iterations, which is set by a human designer.

3.5 Comparison between Conventional LCSs and OCS

In the context of LCS, conventional LCSs can be roughly divided into two types: the Michigan [Holland 78] and Pittsburgh [Smith 83] approaches. As shown in Table 1, the former approach includes many rules in a population, and these rules are evaluated by credit assignment. The latter approach, on the other hand, includes many rule sets in a population, and these rule sets are evaluated through elitist preserving selection [DeJong 75].[7] Compared to these two conventional LCSs, OCS includes many rule sets in a population, but the rules in the rule sets are evaluated by credit assignment. Although other differences between conventional LCSs and OCS can be found in both genetic operations (crossover, mutation, inversion) and a concept of OL as shown in Table 1, the decisive difference is that (1) OCS has a multiagent learning architecture which

[7] Although several methods for an elitist preserving selection are proposed, the method in this paper just means to remove rule sets with a bad evaluation by introducing GA operated rule sets with a good evaluation.

Table 1. Comparison of characteristics between conventional LCSs and OCS

	Michigan	Pittsburgh	OCS
Population	rule	rule set	rule set
Evaluation	rule	rule set	rule
	credit assignment	elitist preserving selection	credit assignment
Crossover	between rules	between rule sets	between rule sets
Mutation	rule	rule	—
Inversion	—	rule set	—
Concept of OL	a part *	a part *	whole
Problem solving	single agent type	single agent type	multiagent type
	a LCS solves a problem	LCSs solve their own problems	LCSs solve a problem together

* Detail analysis is discussed in Sect. 6.

```
procedure OCS
  begin
    iteration=0;
    organizational knowledge reuse;
    while iteration < max iteration do
      begin
        step=0;
        while solution does not converge do
          begin
            rule generation;
            rule exchange;
            step=step+1;
          end
        iteration=iteration+1;
        reinforcement learning;
        organizational knowledge reuse;
      end
  end
```

Fig. 5. Algorithm of OCS

solves given problems through the cooperation of multiple agents, and (2) OCS addresses the role specialization in multiple agents. Both issues are difficult to achieve in the framework of conventional approaches.

4 Pentomino Tiling Problem

4.1 Problem Description

A pentomino is a figure that combines five squares as shown in Fig. 6 (a), and its tiling problem is to appropriately place the pentominos while minimizing the

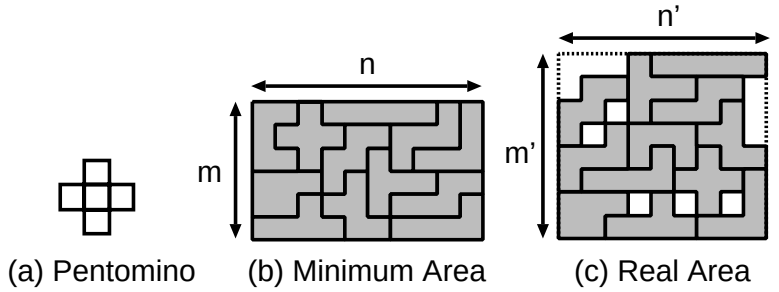

Fig. 6. Pentominos

area that encloses all of the pentominos without any overlap. We select this domain because (1) this problem can be considered as a multiagent problem when one pentomino is assumed to be one agent; (2) it is easy to generate POMDP and NOMDP environments by preparing rules which include local environmental information without other agents information; (3) the minimum solution is known as shown in Fig. 6 (b); and (4) this problem can be directly applied to engineering domains such as printed circuit boards (PCBs) design problems (in this case, each part corresponds to each pentomino [Takadama 99]).

4.2 Pentomino Design and Problem Setting

In the task, each pentomino is designed as an agent in OCS and learns to acquire an appropriate sequence of actions that minimizes the area enclosing all of the pentominos without any overlap. In detail, the pentominos have 17 primitive actions including stay, move, and rotate, and the pentominos get a turn in order, to execute such actions. In a concrete problem setting, all of the pentominos are initially placed at random without considering any overlap, and thus most of the pentominos actually overlap with others. After this initial placement, the pentominos start to perform some primitive actions to reduce such overlap while minimizing the area that encloses all of them. When the size of this area converges without any overlap, all of the pentominos evaluate their own sequences of actions according to the size of the area. Then, the pentominos restart from the initial placement to acquire more appropriate sequences of actions to find a smaller area. In this cycle, one *step* is counted when all of the pentominos perform one primitive action, and one *iteration* is counted when the size of the area converges without overlap. Note that the size of the area enclosing all of the pentominos at each iteration always converges, because its value finally gets into a local minimum when all of the pentominos can no longer find new locations of smaller area than in the current situation without breaking the non-overlap situation.

4.3 Index of Evaluation

The following two indexes are employed as evaluation criteria in the task, and the *performance* in this paper is defined as a criterion which considers the two indexes.

$$Solution = (Real\ area)/(Minimum\ area) \tag{1}$$

$$Computational\ cost = \sum_{i=1}^{n} step(i) \tag{2}$$

The first index (*solution*) evaluates the area enclosing all pentominos and shows how the current area, like that shown in Fig. 6 (c), is small compared with the minimum area in Fig. 6 (b). The next index (*computational cost*) calculates the accumulated steps. Especially in the latter equation, "*step* (i)" and "n" indicate the steps counted until the size of the area converges in i iterations, and the maximum number of iterations, respectively.

Note that real computational cost must include all content shown in Table 2. For example, when $cost1(i)$, $cost2(i)$, and $cost3(i)$ in OCS are respectively assumed as the cost of every step, the cost of every `GA_STEP`, and the cost of every iteration, the three costs in an i iteration are calculated as follows.

$$cost1(i) = n_a s(i) \cdot (n_r lc + k_a c)$$
$$cost2(i) = n_a \frac{s(i)}{\texttt{GA\_STEP}} \cdot (n_r glc)$$
$$cost3(i) = n_a (s(i) \cdot c + n_r lc)$$

In the above equations, n_a, $s(i)$, n_r, l, c, k_a, and g indicate the number of agents, the number of steps in an i iteration, the number of rules, the length of a rule, the cost for one value (or bit) match, the coefficient that adjusts the cost for

Table 2. Real computational cost

	Michigan	Pittsburgh	OCS
Every step	n_a agents* perform one action	pn_a agents* perform one action	n_a agents* perform one action
Every `GA_STEP`†	crossover + mutation	—	crossover
Every iteration	credit assignment	crossover + mutation + inversion	credit assignment + organizational knowledge

* n_a indicates the number of agents in an organization.
p indicates the number of LCSs that solve the same problem at the same iteration.
† `GA_STEP` in Pittsburgh approach can be considered as one iteration.

an action ($k_a >> 1$: In general, the cost of performing an action is much larger than c), and the generation gap ($0 < g < 1$), respectively. In particular, the first and second terms in $cost1$ indicate the cost of rule matching and the cost of performing an action, respectively. In $cost3$, the first and second terms indicate the cost of a credit assignment and the cost of storing organizational knowledge, respectively. These equations derive a relationship $cost1(i) >> \{cost2(i), cost3(i)\}$ that is also satisfied in both the Michigan and Pittsburgh approaches. Thus, the computational cost in an i iteration can be calculated as follows.

$$\begin{aligned} Computational\ cost(i) &= Cost1(i) + Cost2(i) + Cost3(i) \\ &\approx Cost1(i) \\ &= C \cdot s(i) \end{aligned}$$

In this equation, C indicates a constant number represented by $n_a(n_r lc + k_a c)$. From this equation, this paper employs the Eq. (2) as the computational cost. Note that the computational cost of the Pittsburgh approach must be calculated as $p \cdot s(i)$ instead of $s(i)$, because the p number of LCSs solve the same problem at the same iteration.

4.4 Rule Set Design

The classifiers (CFs) in the rule sets of the pentominos are designed as follows for the pentomino tiling problem. These rules are use both for individual and organizational knowledge.

- **The condition part**
  - A previously performed primitive action (the 17 types described in the previous section);
  - A flag distinguishing whether a pentomino overlaps with others or not (1 or 0);
  - A flag distinguishing whether a pentomino is totally adjoined to other surrounding pentominos or not (1 or 0); and
  - A flag distinguishing whether a pentomino removes an overlapping area within a certain time or not (1 or 0). This condition only requires two types of memory for both the one past situation and the count of not removing an overlapping area.
- **The action part:** primitive action (17 types).
- **The strength part:** Strength value.

According to this rule design, an example of 2 1 0 # 6 : 0.5 in CF indicates that *if* `a previous action is the 2nd action` and `there is an overlap` and `the pentomino is not totally adjoined`, *then* `employ the 6th action`, with a 0.5 strength value. In this rule, the # mark indicates "don't care."

5 Simulation

5.1 Experiment Design

To explore elements that cope with non-Markov properties in multiagent environments, we conduct a simulation to compare the following three cases in the pentomino tiling problem with 24 pentominos.

- Michigan Approach [Holland 78]
- Pittsburgh Approach [Smith 83]
- OCS

We select the Michigan and Pittsburgh approaches as our targets to compare with OCS, because both approaches are standard in the context of LCS literature and have different aspects from those of OCS.

5.2 Experimental Setup

For the pentomino tiling problem, organizational knowledge and valuables in OCS are designed as follows.

- **Organizational knowledge** in this simulation is a set comprising each pentominos rule set acquired by 12 pentominos in the prior simulations and is reused as the initial rule sets of 24 pentominos. Note that this knowledge is not randomly generated but is acquired by 12 pentominos. Concretely, 24 pentominos utilize the rule sets as follows, where $RS_{24}(x)$ indicates the rule set of the x-th pentomino whose total number is 24.

$$RS_{24}(x) \leftarrow RS_{12}(mod((x-1), 12) + 1),\ x = 1, \cdots, 24$$

  Through this reuse, the role specialization of 12 pentominos is transfered to that of 24 pentominos as a double role specialization.
- **Parameters** in the Michigan approach, the Pittsburgh approach and OCS are shown in Table 3. Note that the tendency of the results does not change drastically with the parameter setting.

5.3 Experimental Results

Figure 7 shows results of both *solution* and *computational cost* in the Michigan approach, the Pittsburgh approach, and OCS. In this experiment, all results are averaged from five situations with different random seeds. The steps needed to acquire the organizational knowledge are added to the results in OCS, and the *computational cost* in the Pittsburgh approach is adjusted by multiplying p, which is the number of LCSs. The reason for the multiplication is because the same problem is solved by each LCS at the same iteration in the Pittsburgh approach. From this figure, we find the following characteristics.

- The computational cost in the Michigan approach is small, but the solution itself is not good.

Table 3. Parameters in Michigan, Pittsburgh, and OCS

	Michigan	Pittsburgh	OCS
Population size	1 LCS	24 LCSs	
FIRST_CF	30		
MAX_CF	50		
GA_STEP	20 steps	1 iteration	20 steps
Crossover	1 point crossover		rule exchange
GENERATION_GAP	0.1		
BORDER_ST	—		–50
Mutation	1 bit change		—
Mutation rate	0.05		—
Inversion rate	—	0.05	—
Credit assignment	profit sharing	—	profit sharing

* FIRST_CF, MAX_CF, GA_STEP, GENERATION_GAP, and BORDER_ST indicate the number of initial rules of each pentomino, the maximum number of rules of each pentomino, the interval steps in GA operations, the ratio of removed rules, and the lowest strength of the rule not for removal, respectively

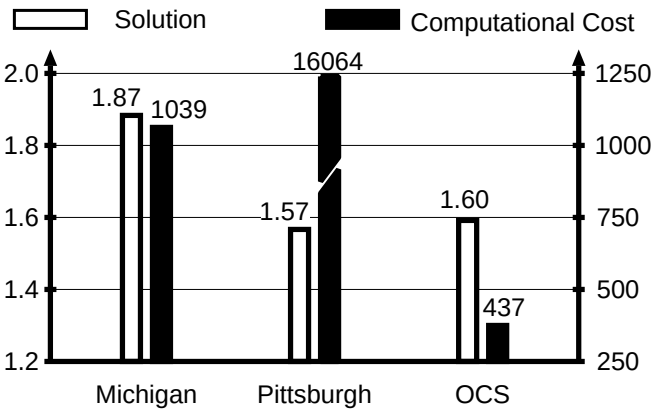

Fig. 7. Solution and computational cost: comparison of Michigan, Pittsburgh and OCS

– The solution in the Pittsburgh approach is quite good, but requires a huge computational cost.
– In comparison with the above results, the solution of OCS is almost the same as that of the Pittsburgh approach, and has the smallest computational cost.

6 Discussion

(1) Analysis from the Viewpoint of OL

As shown in Fig. 7, OCS finds better solutions at smaller computational costs than the Michigan and Pittsburgh approaches. This section investigates why we obtained this result from the viewpoint of OL. We employ the concept of OL as a method for measuring a capability of LCSs towards multiagent environments, because our previous research found that the integration of the four learning mechanisms in OL contributes to finding good solutions at small computational costs in multiagent environments [Takadama 99].

An analysis from the viewpoint of OL tells us that some of the learning mechanisms in OL are missing both in the Michigan and Pittsburgh approaches whereas OCS includes all four of them. Concretely, the Michigan approach includes individual single-loop learning (reinforcement learning) and individual double-loop learning (crossover and mutation),[8] but the other two mechanisms are missing. The Pittsburgh approach, on the other hand, includes individual single- and double-loop learning (inversion and mutation) and organizational single-loop learning (crossover), but lacks organizational double-loop learning. In particular, the lack of the rule exchange mechanism for organizational single-loop learning in the Michigan approach prevents the solution from improving. This is because pentominos in the Michigan approach are limited to independently acquire their own appropriate rules that contribute to finding good solutions. This means that the explored results of other pentominos is required to overcome the limitation of the search range of one pentomino. Furthermore, the lack of the organizational knowledge reuse mechanism for organizational double-loop learning in both the Michigan and Pittsburgh approaches requires large computational costs. This is because pentominos in both the Michigan and Pittsburgh approaches do not originally have an architecture for utilizing pre-learned rules at an organizational level, which contribute to reducing the computational costs.

From this analysis, the learning mechanisms at the *organizational level* are required to overcome limitations in the performance of conventional LCSs in *multiagent environments*. This indicates that such learning mechanisms can be one of the elements to cope with multiagent environments. Here, what we should point to notice is that the organizational level learning mechanisms can maintain their effectiveness owing to the individual level learning mechanisms. For example, the profit sharing as credit assignment procedures of RL in OCS is generally useful only in stationary (*e.g.* single agent) environments, but the rule exchange mechanism and organizational knowledge reuse mechanism in OCS contribute to coping with non-stationary (*e.g.* multiagent) environments by utilizing locally optimized rules acquired by the profit sharing. It is important to utilize locally optimized rules rather than optimize rules in multiagent environments.

[8] The crossover and mutation in the Michigan approach contribute to creating new rules but do not work at an organizational level.

(2) Analysis of Non-Markov Properties

The previous analysis suggests that both organizational single- and double-loop learning are required to improve the performance in multiagent environments. However, this implication is derived only from the viewpoint of OL, and we have not yet understood the relationship between the concept of OL and non-Markov properties. Therefore, this section investigates this relationship to explore elements for coping with multiagent environments.

First, organizational single-loop learning has the capability of estimation of environmental situations at the organizational level. This is because the behavior selection after others' rule acquisition through a rule exchange as organizational single-loop learning can be roughly approximated as behavior selection based on estimation of environmental situation. This means behavior selection based on other agents' internal model (*i.e.*, rule sets in OCS) partly acquired by exchanging rules among agents. Second, organizational double-loop learning, on the other hand, has the capability of utilization of records of past situations/actions at the organizational level. This is because rule sets of all pentominos acquired through organizational double-loop learning can be roughly considered as a condensation of the records on the past situations/actions.

From these facts, both the estimation of environmental situations and the utilization of records of past situations/actions at the *organizational level* have the great potentials to cope with *multiagent environments*. Specifically, an integration of both methods at the *organizational level* has a capability of handling the NOMDP environments, while both methods at the *individual level* cope with the POMDP environments. This indicates that it is quite important to consider such methods at the organizational level to improve the performance in multiagent environments. Furthermore, the NOMDP environments might be effectively addressed by combining methods for the POMDP environments. Moreover, we must not forget that the performance at the organizational level can be improved just by introducing methods at the organizational level, even though they are simple and not elaborated one. In OCS, the rule exchange mechanism and organizational knowledge reuse mechanisms contribute to improving the performance in multiagent environments, even though they are implemented quite simply.

7 Conclusion

This paper focused on non-Markov properties in LCSs and discussed difficulties of addressing the properties in multiagent environments. Towards such difficulties, this paper explored elements for coping with non-Markov properties in multiagent environments by comparing the performance of OCS with those of the conventional LCSs. Although this exploration does not cover entire non-Markov properties in multiagent environments, intensive simulations of the pentomino tiling problem have revealed the following potential implications: (1) OCS finds good solutions at small computational costs in comparison with conventional LCSs, namely the Michigan and Pittsburgh approaches; (2) the learning mechanisms at the organizational level, which means the organizational single- and

double-loop learning mechanisms in OL, contribute to improving the performance in multiagent environments; (3) an estimation of environmental situations and utilization of records of past situations/actions must be implemented at the organizational level to cope with non-Markov properties in multiagent environments.

Future research will include the following: (a) a theoretical analysis of the implications found in this paper; (b) a comparison of the performance of OCS with those of ZCS [Wilson 94], XCS [Wilson 95], and plain GA in multiagent environments; (c) other elements required in LCSs to cope with non-Markov properties in multiagent environments; and (d) categorization of non-Markov properties in multiagent environments.

References

[Argyris 78] C. Argyris and D. A. Schön: *Organizational Learning*, Addison-Wesley, 1978.

[Chirsman 92] L. Chirsman: "Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach", *The 10th National Conference on Artificial Intelligence (AAAI '92)*, pp. 183–188, 1992.

[Cliff 95] D. Cliff and S. Ross: "Adding Temporary Memory to ZCS", *Adaptive Behavior*, Vol. 3, No. 2, pp. 101–150, 1995.

[Cohen 95] M. D. Cohen and L. S. Sproull: *Organizational Learning*, SAGE Publications, 1995.

[DeJong 75] K. Dejong: *An analysis of the behavior of a class of genetic adaptive system*, Ph.D. Thesis, University of Michigan, 1975.

[Goldberg 89] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[Grefenstette 88] J. J. Grefenstette: "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms," *Machine Learning*, Vol. 3. pp. 225–245, 1988.

[Holland 78] J. H. Holland and J. Reitman: "Cognitive Systems Based on Adaptive Algorithms," in D. A. Waterman and F. Hayes-Roth (Eds.), *Pattern Directed Inference System*, Academic Press, 1978.

[Kim 93] D. Kim: "The Link between individual and organizational learning," *Sloan Management Review*, Fall, pp. 37–50, 1993.

[Lanzi 98] P. L. Lanzi: "Adding Memory to XCS", *IEEE International Conference on Evolutionary Computation (ICEC '98)*, pp. 609–614, 1998.

[Lanzi 99] P. L. Lanzi and S. W. Wilson: "Optimal Classifier System Performance in Non-Markov Environments", *technical report N.99.36, Dipartimento di Elettronica e Informazione*, 1999.

[MacCallum 93] R. A. MacCallum: "Overcoming Incomplete Perception with Utile Distinction Memory", *The 10th International Conference on Machine Learning (ICML '93)*, pp. 190–196, 1993.

[Russell 95] S. J. Russell and P. Norving: *Artificial Intelligence: A Modern Approach*, Prentice-Hall International, 1995.

[Smith 83] S. F. Smith: "Flexible learning of problem solving heuristics through adaptive search," *1983 International Joint Conference on Artificial Intelligence (IJCAI '83)*, pp. 422–425, 1983.

[Sutton 98] R. S. Sutton, A. G. Bart: *Reinforcement Learning – An Introduction –*, The MIT Press, 1998.

[Takadama 98] K. Takadama, S. Nakasuka, and T. Terano: "Multiagent Reinforcement Learning with Organizational-Learning Oriented Classifier System," *IEEE 1998 International Conference On Evolutionary Computation (ICEC '98)*, pp. 63–68, 1998.

[Takadama 99] K. Takadama, T. Terano, K. Shimohara, K. Hori, and S. Nakasuka: "Making Organizational Learning Operational: Implications from Learning Classifier System", *Computational and Mathematical Organization Theory (CMOT)*, Kluwer Academic Publishers, Vol. 5, No. 3, pp. 229–252, 1999.

[Weiss 99] G. Weiss: *Multiagent Systems – Modern Approach to Distributed Artificial Intelligence*, The MIT Press, 1999.

[Wilson 94] S. W. Wilson: "ZCS: A Zeroth Level Classifier System", *Evolutionary Computation*, Vol. 2, No. 1, pp. 1–18, 1994.

[Wilson 95] S. W. Wilson: "Classifier Fitness Based on Accuracy", *Evolutionary Computation*, Vol. 3, No. 2, pp. 149–175, 1995.