Focus

Soft Computing 6 (2002) 229 – 239 © Springer-Verlag 2002
DOI 10.1007/s005000100118

Robustness in organizational-learning oriented classifier system

K. Takadama, S. Nakasuka, K. Shimohara

Abstract An organizational-learning oriented classifier system (OCS) is an extension of learning classifier systems (LCSs) to multiagent environments, where the system introduces the concepts of organizational learning (OL) in organization and management science. To investigate the capabilities of OCS as a new multiagent-based LCS architecture, this paper specifically focuses on the *robustness* of OCS in multiagent environments and explores its capability in space shuttle crew task scheduling as one of real-world applications. Intensive simulations on a complex domain problem revealed that OCS has robustness capability in the given problem. Concretely, we found that OCS derives the following implications on robustness: (1) OCS finds good solutions at small computational costs even after anomaly situations occur; and (2) this advantage becomes stronger as the number of anomalies increases.

Keywords Learning classifier system, Multiagent system, Robustness, Crew task scheduling, Organizational learning

1 Introduction

Since Wilson proposed ZCS [26] and XCS [27], research on learning classifier systems (LCSs) has come to be active again.[1] In particular, the capabilities of LCSs are currently being studied from the viewpoints of *rule generalization* [28], *rule chain or linkage* [24], *non-Markov environments* [16], and so on. Furthermore, a LCS has been applied to the robotics area as a controller of a physical robot [8]. In comparison with these research works which have mainly employed a single LCS, there is few research on multiagent-based LCSs. As major examples among the few works, Bull controlled a quadrupedal robot through a coordination of its four legs, where each leg was implemented by a Pittsburgh-style [20] classifier system [3, 4], while Arthur investigated asset prices in an artificial stock market by employing multiple Michigan-style [12, 13] classifier systems as interacting agents [27]. In addition to these two works, others have addressed issues in multiagent systems (MAS) [25] or distributed artificial intelligence (DAI) [9] using LCSs [7, 5, 17, 19]. However, most of all of the researches in this area have focused on their own goals and have analyzed results from their own viewpoints. Accordingly, the capabilities of proposed multiagent-based LCSs have not been investigated consistently and comprehensively. Due to this lack of cumulative processes of proposed architectures, it has been difficult to determine the applicable ranges of these architectures. This has prevented us from employing such architectures for given problems; this is a serious problem from the engineering viewpoint.

To overcome this problem little by little, our previous research not only showed the effectiveness of our multiagent-based LCSs [22] but also investigated their capabilities from several viewpoints [23]. Specifically, the former effectiveness of our LCSs was showed by finding better solutions at smaller computational costs than the human experts in printed circuit board (PCB) re-design problems in the computer aided design (CAD) domain. The latter capabilities of our LCSs, on the other hand, were investigated from the viewpoint of the *generality*, *scalability*, and *performance* towards consistent and comprehensive analysis. This analysis, in particular, was done by applying our LCSs in another domain, investigating their characteristics for large-scale problems, and comparing the performance with that of conventional LCSs, namely the Michigan and Pittsburgh approaches. Note that the comparison with conventional LCSs also includes the comparison of *multiagent* type (our LCSs) of problem solving with both *single* (Michigan approach) and *parallel* (Pittsburgh approach) types of problem solving. From this previous research, we found that our LCSs have the following potential: "generality" to show a good performance in other domains, "scalability" to maintain the same level of performance for large-scale problems, and "a high performance" that is better than that in conventional LCSs.

However, the above effectiveness was shown only from certain aspects and we should therefore analyze our LCSs from several other aspects to determine their applicable ranges precisely. Specifically, we must not forget that investigations on *robustness* still remain as an important and indispensable aspect for engineering problems and this aspect must be considered for real-world applications.

K. Takadama (✉)
ATR International, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
E-mail: keiki@isd.atr.co.jp

S. Nakasuka
University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: nakasuka@space.t.u-tokyo.ac.jp

K. Shimohara
ATR International, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
E-mail: katsu@isd.atr.co.jp

[1] See [15] for recent research on LCSs.

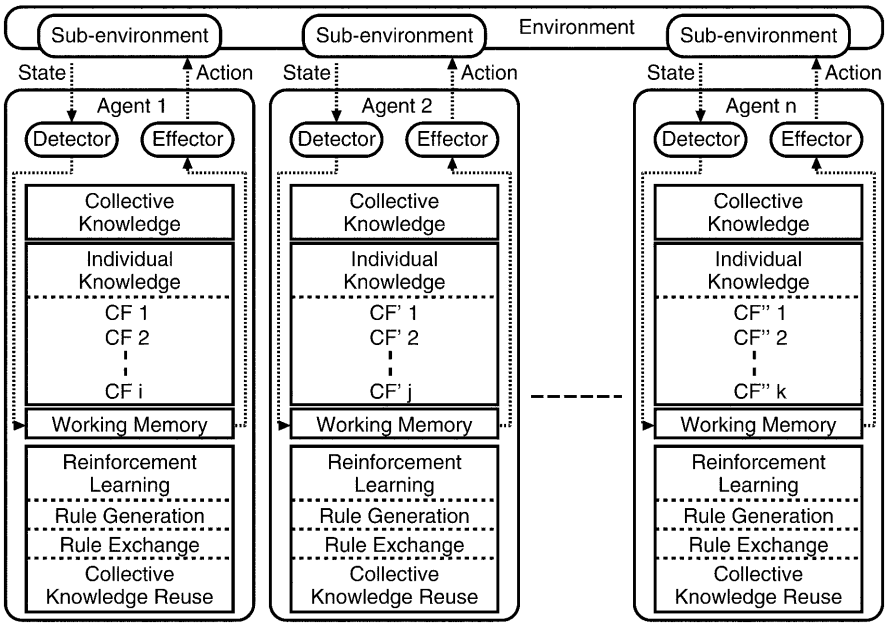

Fig. 1. OCS architecture

From this fact, this paper focuses on the *robustness* of our multiagent-based LCSs and explores the capabilities of our LCSs for space shuttle crew task scheduling as one of real-world applications. We, in particular, employ an *actual* real-world application to investigate applicable ranges of our architecture. Through the experiment, this paper finally makes conclusions about the total capabilities of our LCSs from the perspective of towards widely applicable LCSs in multiagent environments.

This paper is organized as follows. Section 2 describes our multiagent-based LCSs, and Sect. 3 gives an example of a real-world application for analyzing the robustness of our LCSs. Simulation results are given in Sect. 4, and the robustness of our LCSs is discussed from several viewpoints in Sect. 5. Finally, our conclusions are made in Sect. 6.

2
Organizational-learning oriented classifier system

2.1
Outline

An organizational-learning oriented classifier system (OCS) [22] is an extension of learning classifier systems (LCSs) [10] to multiagent architectures that introduce the concept of organizational learning (OL) [1, 6] in organization and management science. Specifically, OCS addresses multiagent environments that include imprecision, uncertainties and partial truths, by complementarily integrating four kinds of learning mechanisms [14] computationally interpreted from OL.

2.2
Agents

In OCS, agents are implemented by their own LCSs, which are extended to introduce the 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 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 the 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.

2.3
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 [18], respectively.

Memory

- **Collective knowledge memory** stores a set comprising each agent's rule set as collective knowledge. In OCS, this knowledge is shared by all agents and represents knowledge on role specializations.
- **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 each 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 information for classifiers.[3] OCS acquires such information through recognition

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

[3] Since this information depends on the given problem, the concrete information in this paper is described in Sect. 3.4.

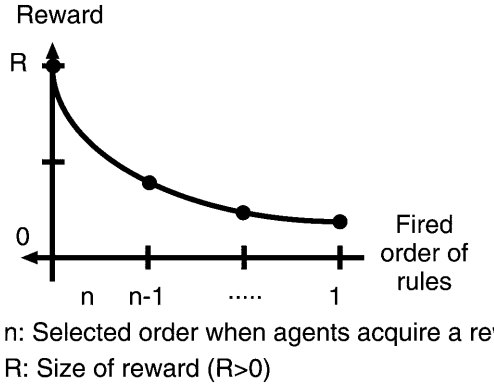

Fig. 2. Reinforcement learning

of sub-environmental results and performs some actions according to fired rules that reflect the information.

Mechanisms

- **Reinforcement learning mechanism, rule generation mechanism, rule exchange mechanism, and collective knowledge reuse mechanism** are computationally interpreted from the four kinds of learning in OL (A brief explanation is given later[4]). These mechanisms are not improved by specific or elaborate techniques but by simple and ordinary ones.

2.4 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. To implement this mechanism, OCS employs a *profit sharing* method [11] that reinforces the sequence of all rules when agents obtain some rewards. Brief explanation of RL in OCS is described in Fig. 6-1.

As a concrete mechanism, the strength values of all fired rules are changed through a distribution of positive rewards shown in Fig. 2, and these strength values are calculated according to Eq. (1).[5]

$$ST(i) = ST(i) + R \cdot G^{n-i}, \quad \text{where } i = 1, 2, \ldots, n \qquad (1)$$

In Fig. 2, the vertical and horizontal axes indicate the size of the reward and the fired order of rules, respectively. Note that the rules presented on the right side are fired at the first several selections. Furthermore, $ST, i, n, R(>0), G$ in the equation indicate the strength value of the rule, the order of the fired rules, the total number of fired rules

[4] See [22] for a detailed interpretation and implementation of the four kinds of learning in OL.

[5] The detailed explanation of this way of credit assignment in OCS was described in [21].

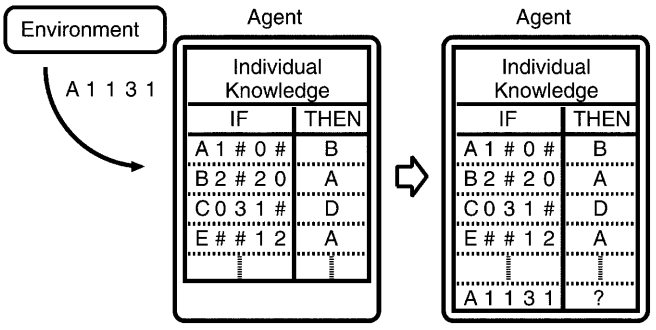

Fig. 3. Rule generation

when solving given problems, the size of the reward, and the geometric ratio with the range of $0 < G < 1$, respectively. A small i indicates the first several fired order.

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. 6-2. 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. For example, when there is no rules which match to "A1131" decoded as an environmental state, new rule is created by setting "A1131" in the condition part, determining at random in the action part, and setting the initial value in strength value as shown in Fig. 3. Note that the rule with the lowest strength is removed and a new rule is generated, when the number of rules is `MAX_CF` (maximum number of rules).

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 incremented when the No. i rule is fired and is reset to 0 when the situation changes. With this mechanism, the strength values of fired rules decrease as long as the situation does not change as in deadlocked situations where the same rules are selected repeatedly. These rules become candidates capable of being replaced by new rules, while the strength values of these rules recover.

3. Rule exchange mechanism

In OCS, agents exchange rules with other agents at particular time intervals (`EXCHANGE_STEP`[6]) in order to solve given problems that cannot be solved at an individual level. In this mechanism, a particular number (the number of rules $\times$ `GENERATION_GAP`[7]) of rules with low strength values is replaced by rules with high strength values between two arbitrary agents. This kind of rule exchange is done by all pairs of agents.

For example, when agents X and Y are selected as one pair of agents as shown in Fig. 4, CFs in agents X and Y are

[6] This step is defined in Sect. 3.2.

[7] The ratio of removed rules.

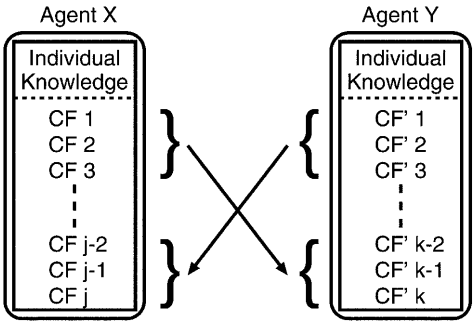

Fig. 4. Rule exchange

sorted by order of their strength values (upper CFs have high strength values). After this sorting, $CF_{j-2} \sim CF_j$ and $CF'_{k-2} \sim CF'_k$ become a candidate of being removed when (the number of rules) $\times$ GENERATION_GAP is 3, and these CFs are respectively replaced by $CF'_1 \sim CF'_3$ and $CF_1 \sim CF_3$, if the strength values of $CF_{j-2} \sim CF_j$ and $CF'_{k-2} \sim CF'_k$ is lower than a particular value (BORDER_ST). This kind of rule exchange contributes not only to exchanging good rules among agents but also to avoiding unnecessary rule exchanges. However, 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. Brief explanation of rule exchange in OCS is described in Fig. 6-3.

4. Collective knowledge reuse mechanism

Finally, agents in OCS store a set comprising each agent's rule set (individual knowledge) as knowledge on role specializations, when they most effectively solve given problems.[8] After storing this knowledge, agents can reuse it for future problem solving, instead of using initial individual knowledge generated at random. This mechanism enables agents to use the best knowledge acquired previously. We refer to the collective knowledge as knowledge on role specializations, because the collective 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 collective level.

The concrete mechanism of collective knowledge reuse in OCS is as shown in Fig. 6-4. For example, when we consider n agents address a given problem, agents reuse the collective knowledge (i.e., a set comprising each agent's rule set) as the initial rule sets, if the collective knowledge is already stored. Then, agents store other collective knowledge when they solve their problem most effectively, and update it through problem solving. In this case, the collective knowledge already stored are replaced by new ones. As shown in Fig. 5, this knowledge is shared among agents and is represented by $\{\mathrm{RS}\ (1), \mathrm{RS}\ (2), \ldots, \mathrm{RS}\ (n)\}$, where RS ($x$) is the rule set for the x-th agent. This design indicates that each agent does not store the complete individual rule sets of all of the other agents independently but shares the rule sets of all agents with other agents.

Some other characteristics of the collective knowledge are summarized as follows:

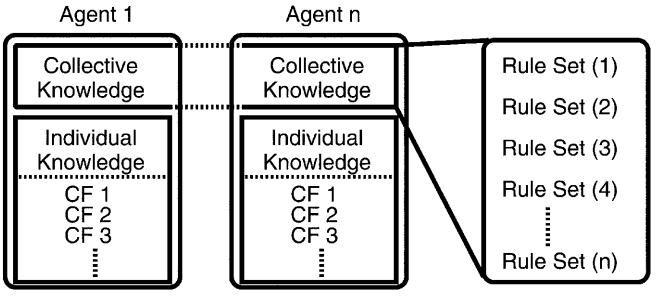

Fig. 5. Collective knowledge reuse

```
procedure reinforcement learning
  begin
    if solution is converged then
      for all agents do
        for all fired rules do
          the strength value of each fired rule is added according
            to the positive rewards;
  end
```

Figure 6–1

```
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;
          the strength value of the new rule is set to the initial one;
        end
  end
```

Figure 6–2

```
procedure rule exchange
  begin
    if mod (step, EXCHANGE_STEP)=0 then
      for all pairs of agents do
        for (number of rules)×GENERATION_GAP rules do
          if the lowest strength value of a 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;
              the strength value of the replaced rule is reset to its
                initial value;
            end
  end
```

Figure 6–3

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

Figure 6–4

Fig. 6. Algorithms of four learning mechanisms

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

- The collective knowledge is different from the ordinary effective knowledge in a single LCS, because the former knowledge represents role specializations and is used in the unit of *multiple agents*, while the latter knowledge is utilized in the unit of *one agent*. Note that the collective 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 collective knowledge, i.e., knowledge that the same agent or another agent acquired in problem solving previously.
- Agents cannot use both individual and collective 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 agents *after* or *before* problem solving.

2.5 Relationships among the four learning mechanisms

The total algorithm of OCS follows the procedure shown in Fig. 7. Briefly, collective 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 converges. After the convergence of the solution, both the reinforcement learning and collective 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.

```
procedure OCS
  begin
    iteration=0;
    collective 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;
        collective knowledge reuse;
      end
  end
```

Fig. 7. Algorithm of OCS

3 Crew task scheduling

3.1 Problem description

Crew task scheduling for a space shuttle or a station is a job-shop scheduling problem where many jobs of the crew members must be scheduled under hard resource constraints. The goal of this problem is to find feasible schedules that minimize the total schedule execution (TSE) time of all jobs. We selected this domain because (1) this problem is an *actual* real-world application and the system for this problem is really required to be implemented at ground stations to support human schedulers; (2) we can consider this problem as a multiagent problem when one job is assumed to be one agent; and (3) robustness can be measured through a lot of unexpected anomalies that often occur.

As a concrete problem, there are two types of crew members and a number of jobs, as indicated by the boxes, in Fig. 8. Specifically, the type of crew is divided into *mission specialists* (MSs) and *payload specialists* (PSs), which are mainly in charge of experiments and support experiments, respectively. The jobs, on the other hand, belong to the missions of the crew members (this means that each mission is composed of some jobs) and have the following pre-determined information: (a) job processing time; (b) required number of crew members; and (c) necessary types of crew members. Based on such information, all jobs should be assigned to the appropriate crew members while satisfying the following constraints to accomplish the mission.

1. Power of space shuttle or station: Each job requires a particular power (0–100%) during experiments, but the summation of the power involving all jobs at any time

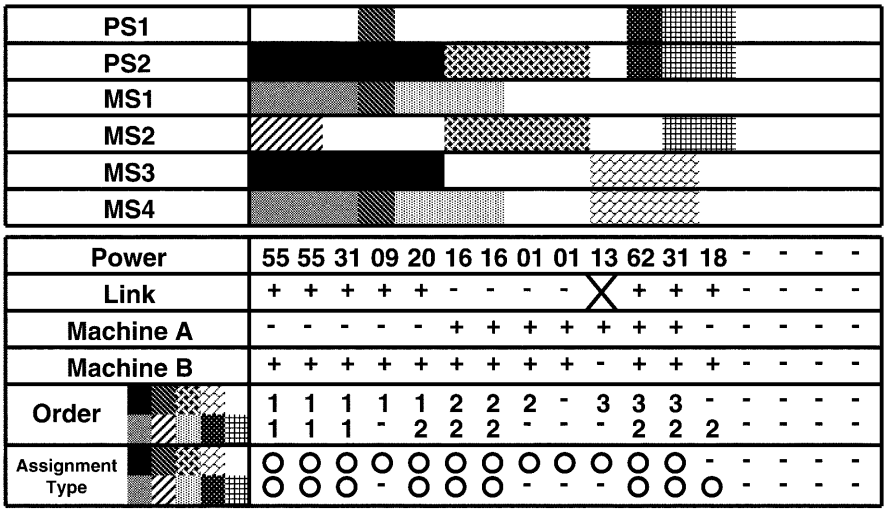

Fig. 8. Crew task scheduling

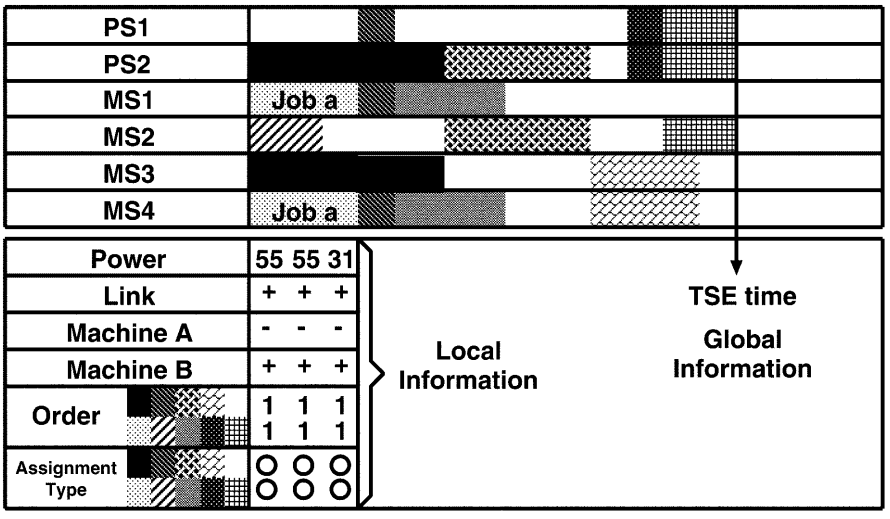

Fig. 9. Local and global information for each Job

must not exceed 100%. For example, the power at the first unit schedule time is 55% in Fig. 8.

2. Link to the ground station: Some jobs require a link to the ground station, but only one job at an unit schedule time can use the link. Due to the orbit of the spacecraft, none of the jobs can use the link during certain times. In Fig. 8, a plus mark indicates that one job is utilizing the link while a minus mark indicates that none of the jobs is utilizing it. A cross mark in this constraint indicates a point in time where jobs can not use the link.

3. Machine A: Some jobs need to use machine A during experiments, but only one job at a time can use machine A. Some examples of machines are computers, voice recorders, and so on. The meanings of the plus and minus marks are the same as those for the link constraint.

4. Machine B: This is the same constraint as that for machine A.

5. Execution order of jobs: In a mission unit, jobs have an execution order, but some jobs in a mission may only be partially ordered, which means that some jobs may have the same order.[9] In comparison with jobs, there is no order/priority among missions. For example, the mission located at the upper part of the execution order in Fig. 8 is composed of four jobs and each job has its own execution order from 1 to 3, where a smaller number means a higher order. This example indicates that some jobs have the same execution order.

6. Crew assignment types: In a unit of a job, one of the following crew assignment types must be satisfied: (a) Anybody, (b) PS only (but the concrete crew member is not specified), (c) one specified PS with somebody, (d) one specified MS with somebody, and (e) a combination of PS and MS (but the concrete crew members are not specified). These crew assignments are based on actual space shuttle missions. In Fig. 8, a job indicated as a black box is assigned to PS2 and MS3 to satisfy the constraint of the combination of PS and MS. A circle in this figure indicates the satisfaction of such constraints.

[9] The order described here is the most simple one. Other representations are required to handle partially orders in general.

3.2 Problem setting

To address such crew task scheduling from a multiagent-based viewpoint, each job is designed as an agent in OCS, and each job learns to acquire an appropriate sequence of actions that minimizes the TSE time. Specifically, jobs are designed only to observe local situations in the range of the job processing time except for the TSE time calculated from the location of the job set at the latest time in the schedule. The local information includes information on overlaps, power, links, and so on, while the global information, i.e., the TSE time, is updated synchronously every time after all jobs perform one action such as a movement of their location to an earlier or later schedule time. For example, the *job a* in Fig. 9 can only observe local situations from one to three unit schedule times except for the TSE time. Based on such local observations, only related (neighbor) jobs can recognize changes made by other jobs.

For a concrete problem-solving approach, jobs are designed to have seven kinds of actions. These actions include a movement that reduces overlapping areas (one type) and a movement that satisfies each of the six constraints described in the previous section (six types). One example of the latter movement is a movement that satisfies power constraints. In these actions, the movement is defined as a location change of a job to one unit earlier or later schedule time, when overlapping areas or the six constraints can be reduced or satisfied more than the current location. In particular, the job moves to an earlier schedule time if the degree of reducing overlapping areas and satisfying constraints is the same. This degree is calculated for all types of movements by changing the crew assignment for a job not only in the current location but also in the location of one unit earlier or later schedule time.

Using such actions, all jobs follow the following procedure in the framework of OCS.

1. First, all jobs reuse the collective knowledge as an initial rule set through the *collective knowledge reuse mechanism* when anomalies[10] occur. This knowledge is

[10] Concrete anomalies are described in Sect. 4.1.

pre-generated as a set comprising each job's rule set through a simulation of the normal case (i.e., the case without anomalies). Note that there is no collective knowledge in the normal case.

2. Next, an initial placement of all jobs is determined as follows.

- In the normal case, all jobs are initially placed at random without considering any overlaps or the six constraints described in the previous section.
- In the anomaly case, all jobs are initially placed at random in the same way as the normal case or at placements of final feasible schedules before anomalies occur.[11]

Due to the random placements or anomalies, a schedule is not feasible at this time. This means that some jobs actually come to overlap with others or have unsatisfied constraint situations. Here, we define *initial placements* as the placements where all jobs are initially placed at random and *anomaly introduced placements* as the placements where anomalies occurs in final feasible schedules.

3. After this placement, the jobs get a chance, in order, to change their locations to reduce overlapping areas and to satisfy the constraints while minimizing the TSE time. To reduce such overlapping areas or satisfy the constraints effectively, the jobs acquire appropriate if-then rules through the *rule generation and exchange mechanisms*.

4. When the value of the TSE time converges with a feasible schedule, all of the jobs evaluate their own sequences of actions according to the value of the TSE time. Concretely, the sequences of the fired if-then rules deriving such actions are evaluated by the *reinforcement learning mechanism*.

5. Goto 3 to make all jobs restart from the same initial placement or anomaly introduced placement. Then, all jobs try to acquire more appropriate sequences of actions that find shorter times than in the current situation.

In the above cycle, one *step* is counted when all jobs perform one action driven by a fired rule, and one *iteration* is counted when the value of the TSE time converges with a feasible schedule. Note that the TSE time at each iteration always converges, because its value finally gets into a local minimum when all jobs can no longer find new locations of a shorter TSE time than in the current situation without breaking the non-overlap situation or satisfied constraints.

3.3 Evaluation criteria

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

$$\text{Solution} = TSE \text{ time} \tag{2}$$

$$\text{Computational cost} = \sum_{i=1}^{n} \text{step}(i) \tag{3}$$

[11] Details are described in Sect. 4.1.

The first index (*solution*) evaluates the execution time of a feasible schedule, and the second index (*computational cost*) calculates the accumulated steps. In the latter equation, in particular, "*step* (i)" and "n" indicate the steps counted until the value of the TSE time converges in i iterations, and the maximum number of iterations, respectively.

3.4 Rule set design

Since jobs can only observe local situations, the rules for each job are designed to only consider local information. Note that the conditions of classifier (CF) from 2 to 7 are described in the previous section.

- The condition (IF) part has the following parts:
  1. A flag distinguishing whether a job is overlapped or not (1 or 0);
  2. A flag distinguishing whether the power is over 100% or not at a job's location (1 or 0);
  3. A flag distinguishing whether a link is required by more than two jobs or not at a job's location (1 or 0);
  4. A flag distinguishing whether the machine A is required by more than two jobs or not at a job's location (1 or 0);
  5. A flag distinguishing whether the machine B is required by more than two jobs or not at a job's location (1 or 0);
  6. A flag distinguishing whether a job is located at an appropriate execution order or not (1 or 0);
  7. A flag distinguishing whether a job satisfies crew assignment conditions or not (1 or 0); and
  8. A flag distinguishing whether a situation on overlapping or constraints in a job is the same at a certain time or not (1 or 0). This condition requires working memory for storing one past situation and the count of not reducing an overlapping area or not satisfying constraints.
- The action (THEN) part represents one of the actions (seven types) described in the previous section;
- The strength part is the worth or weight of rules.

According to this design, one example of a classifier (#0 1 # # 0 1 0 | 2 : 0.5) represents that *if* the power is not over 100%, a link is required by more than two jobs, a job is not located at an appropriate execution order, a job satisfies crew assignment conditions, and a situation on overlapping or constraints is not the same at a certain time, *then* employ the 2nd action, with a 0.5 strength value. In this rule, the #mark indicates "don't care."

4 Simulation

4.1 Experimental design

In the following, a simulation investigates the robustness of OCS by introducing anomalies in the crew task scheduling problem. Specifically, we consider the 16 types of anomalies shown in Table 1 and introduce these anomalies into crew task schedules that involve 10 jobs (agents) in five missions. Note that all of the anomalies in

Table 1. Types of anomalies

Type	Anomaly	Content
1	Crew sick	A crew member cannot perform his/her job
2	Power down	The max power capacity decreases
3	Link down	A link cannot be used
4	Machine A down	Machine A cannot be used
5	Machine B down	Machine B cannot be used
6–15	Three combinations of types 1, 2, 3, 4, and 5	Integration of three anomalies
16	Five combinations of types 1, 2, 3, 4, and 5	Integration of five anomalies

this simulation are assumed to be normal after a certain duration, because feasible schedules cannot be found unless anomalies are removed. For example, a job that requires a link cannot be completed as long as such a link is down. From this fact, we pre-determine when each anomaly starts and ends.

In this paper, we compare results from *initial placements* of jobs with those from *anomaly introduced placements* of jobs. In particular, the latter placements means placements of jobs where anomalies occur in final feasible schedules. More precisely, the difference between the initial and anomaly introduced placements is summarized as follows: (a) when starting from initial placements, all jobs are placed at random locations and then move their locations by considering anomaly parts (this corresponds to a reschedule from the beginning); (b) when starting from anomaly introduced placements, on the other hand, all jobs move their locations from a placement where the anomaly parts are introduced in a final feasible schedule (this corresponds to a reschedule from the current schedule).

4.2 Experimental setup

The following components are designed for the crew task scheduling problem.

- **Collective knowledge** in this simulation is pre-generated in the normal case and this knowledge is reused as the initial rule set in the anomaly case. In other words, all jobs utilize their own learned rule sets acquired in the normal case, when anomalies occur.
- **Parameters in OCS** are set as follows. Note that preliminary simulations confirmed that the tendency in the results does not drastically change according to the parameter settings.
  - `FIRST_RULE` (the number of initial rules): 25
  - `MAX_RULE` (the maximum number of rules): 50
  - `RULE_EXCHANGE` (the interval steps for rule exchange operations): 10
  - `GENERATION_GAP` (the percentage of removed rules): 10%
  - `BORDER_ST` (the lowest strength of a rule not for removal): -50.0
  - R (the size of the reward): 1
  - G (the geometric ratio): 0.5

4.3 Experimental results

Table 2 shows solutions (i.e., the TSE time) and computational costs (i.e., the accumulated steps), both of which are calculated from initial placements and from anomaly introduced placements in the crew task scheduling problem. Specifically, Table 2a–c show the results of a single anomaly, three combinations of anomalies, and five combinations of anomalies, respectively. Note that the numbers in parenthesis shown in Table 2b and c, in particular, indicate the combinations of anomalies. For example, the $(1 + 2 + 3)$ in type 6 denotes that type 6 includes three anomalies of types 1, 2, and 3. Furthermore, all of the results in these simulations are averaged from five different random seeds.

From these results, we find the following implications: (1) solutions from anomaly introduced placements are mostly the same as those from initial placements, while computational costs from anomaly introduced placements are quite smaller than those from initial placements; and (2) this tendency is maintained even when the number of anomalies increases.

5 Discussion

5.1 Robustness: good solutions at small computational costs

To analyze the robustness of OCS in more detail, Fig. 10 shows a summarization of the simulation results shown in Table 2. In particular, Fig. 10a and b indicate averages of *solutions* and *computational costs*, respectively. Note that the solutions mean the TSE (total schedule execution) time of all jobs and the computational costs mean the accumulated steps of all jobs, which are calculated by the Eq. (3) described in Sect. 3.3. These solutions and computational costs for one, three, and five anomalies are respectively averaged from five, ten, and one kinds of anomaly types according to the number of anomaly types in Table 2. In these figures, the left axis indicates solutions and computational costs, while the horizontal axis indicates the number of anomalies introduced into OCS. Furthermore, the white and black boxes indicate results from initial placements and those from anomaly introduced placements, respectively.

From the results shown in Fig. 10, we find that OCS which starts from anomaly introduced placements finds good solutions at small computational costs even in anomaly cases. This is one of the great advantages of OCS, and this can be understood more deeply by considering the following implications: (1) it is generally difficult to find good solutions when starting from anomaly introduced placements, because solutions in this case tend to fall into local minima quickly due to an only modification of anomaly parts. Additionally, (2) high computational costs are generally required to get out of local minimum solutions, if we seek good solutions even when starting from anomaly introduced placements. This extension is done by breaking a lot of satisfied constraints.

The above advantage of OCS suggests that OCS has robustness in terms of finding *good solutions* at *small*

Table 2. Solutions and computational costs

Type		One anomaly (5 kinds)					Three anomalies (10 kinds)										Five anomalies (1 kind)
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
							(1 + 2 + 3)	(1 + 2 + 4)	(1 + 2 + 5)	(1 + 3 + 4)	(1 + 3 + 5)	(1 + 4 + 5)	(2 + 3 + 4)	(2 + 3 + 5)	(2 + 4 + 5)	(3 + 4 + 5)	(1 + 2 + 3 + 4 + 5)
From initial placement	TSE time	27.6	30.0	30.0	31.0	29.0	31.8	34.3	31.3	33.5	31.5	30.0	35.0	34.0	30.8	30.5	35.6
	Acc. steps	241.2	672.0	889.3	241.0	1116.2	238.3	1376.0	1377.3	2053.0	319.0	5977.7	834.0	1290.67	5818.0	1242.5	3204.2
From anomaly introduced placement	TSE time	30.2	33.6	33.8	32.4	29.4	34.4	32.8	31.2	36.0	31.6	33.0	35.4	36.2	35.6	36.0	37.2
	Acc. steps	14.0	11.8	43.4	16.4	13.8	77.2	23.4	37.2	35.8	13.8	36.8	19.0	37.22	43.3	16.0	38.0

Acc. steps means accumulated steps

computational costs in anomaly cases. This indicates that OCS can cope with anomaly situation from anomaly introduced placements as continuous problem solving, instead of finding new feasible solutions from initial placements as initial problem solving.

5.2 Robustness: anomaly scale

Next, to investigate the above robustness of OCS from the viewpoint of an anomaly scale, we calculate *differences* between results from initial placements and those from anomaly introduced placements in Fig. 11. In this figure, the left and right axes respectively indicate the difference of solutions and the difference of computational costs, while the horizontal axis indicates the number of anomalies introduced into OCS. In particular, both differences are calculated by |(solutions/costs from the initial placement) − (solutions/costs from anomaly introduced

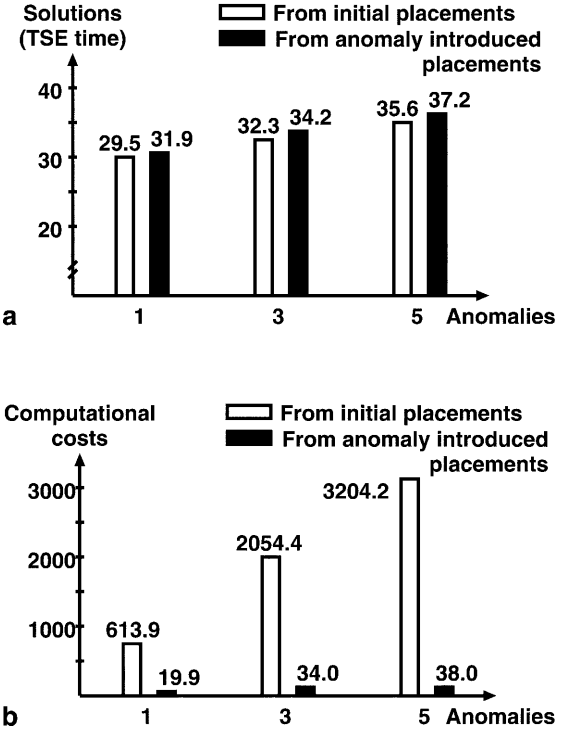

Fig. 10. Results for one, three, and five anomalies

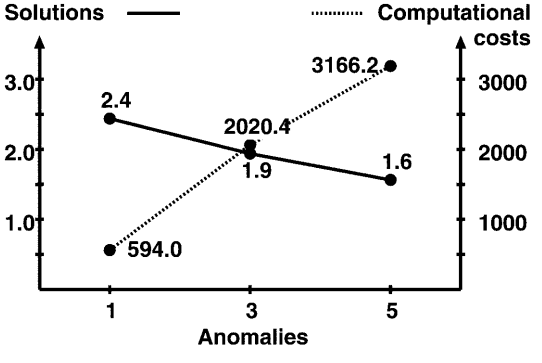

Fig. 11. Difference between results from initial placements and those from anomaly introduced placement

placement)|. Furthermore, the solid and dotted lines indicate the difference of solutions and the difference of computational costs, respectively.

From the results shown in Fig. 11, we find that the differences in solutions and computational costs respectively become small and large as the number of anomalies increases. This clearly indicates that the robustness of OCS becomes stronger as the number of anomalies increase. This is because the following implications are quite important for robustness from the viewpoint of an anomaly scale: (1) solutions from anomaly introduced placements as continuous problem solving become close to those from initial placements as initial problem solving; (2) computational costs from anomaly introduced placements are maintained as small while those from initial placements become large. This advantage of OCS suggests that OCS has robustness in terms of an *anomaly scale*.

5.3 Robustness of OCS

From the above discussion, OCS has robustness in terms of finding good solutions at small computational costs in anomaly cases and such robustness becomes strong in terms of an anomaly scale. So, why does OCS obtain such robustness? The main reasons can be roughly summarized that OCS maintains an appropriate balance between *exploration* and *exploitation* even in anomaly cases. To understand the above reason, let's move our focus to the normal case for a moment. In this case, our previous research revealed that OCS maintains the above balance appropriately as follows: (1) both the *reinforcement learning* and *rule generation* mechanisms mainly explore a solution space to find good solutions; and (2) both the *rule exchange* and *collective knowledge reuse* mechanisms mainly exploit effective rules to reduce computational costs [22].

Moving back our focus to the anomaly cases, an analysis of results shown in Fig. 10 tells us that some of the effective rules acquired by the rule exchange and collective knowledge reuse mechanisms may no longer be effective when anomalies occur. Furthermore, such ineffective rules lead new situations that are not mostly necessary be encountered by jobs for finding feasible schedules. Since this tendency is strong when jobs start from initial placements, high computational costs are required to acquire rules that get out of new unnecessary situations. This can be understood from the consideration that a lot of jobs starting from initial placements change their locations in a schedule to satisfy constraints. To the contrary, good solutions can be found through a wide exploration carried out by searching from the initial placements.

Compared with this type of starting, jobs starting from anomaly introduced placements mostly utilize the acquired rules effectively, because their situations are not so much different from situations before anomalies occur in feasible schedules. Since this means that only jobs which constraints are violated by anomalies change their locations, computational costs are kept in low. Solutions in this case, on the other hand, are slightly worse than those in the case from initial placements. This is because reinforcement learning and rule generation mechanisms are limited to exhibit their abilities in a narrow exploration restrained by searching from the anomaly introduced placements.

From the viewpoint of an anomaly scale, as another important factor, Fig. 11 indicates that the difference between computational costs from initial placements and those from anomaly introduced placements becomes large as the number of anomalies increases, while the opposite tendency is found in the case of solutions. The former, i.e., the enlargement of difference in computational costs, is because jobs increase the case to encounter new situations that are not necessary for finding feasible schedules according to the increase of ineffective rules caused by the anomalies. The latter, i.e., the reduction of difference in solutions, on the other hand, is because the exploration range from anomaly introduced placements is close to the range from initial placements according to the increase of the anomalies.

From these analyses, we have found the following implications: (1) in the case of *small* number of anomalies, an *exploitation* factor (i.e., rule exchange and collective knowledge reuse mechanisms) does not work well but an *exploration* factor (i.e., reinforcement learning and rule generation mechanisms) works well when starting from initial placements while the opposite implications are found when starting from anomalies introduced placements; and (2) in the case of *large* number of anomalies, the implications which are the same in the case of small number of anomalies are found when starting from initial placements, while an *exploitation* factor works well and an *exploration* factor is improved when starting from anomalies introduced placements. This indicates that it is important to search from anomalies introduced placements with an increase the number of anomalies to maintain an appropriate balance between *exploration* and *exploitation*, because such appropriate balance contributes to finding good solutions at small computational costs even in anomaly case. This also indicates that OCS is able to benefit from previously learned knowledge even when constraints are changed/intensified, i.e., in a related but more difficult task, if agents (jobs) start from situations where all constraints are satisfied except for the change or intensification parts. From these discussions, we arrived at the conclusion that OCS has robustness in anomaly cases by maintaining an appropriate balance between exploration and exploitation. Although this indicates that OCS has a small limitation in robustness in the case of inappropriate balance (i.e., the case starting from initial placement or the case of small number of anomalies), the results in this paper show that OCS can offer high applicability for tasks in which unexpected anomalies often occur like in space applications. Such robustness is quite important and indispensable for practical and engineering uses.

6 Conclusion

This paper focused on the *robustness* of OCS in multiagent environments and explored its capability in space shuttle crew task scheduling as one of real-world applications. Although the simulation results in this paper do not cover

all types of anomalies, we arrived at the conclusion that OCS has the robustness in anomaly cases. Concretely, we found that OCS derives the following implications on robustness: (1) OCS finds good solutions at small computational costs even after anomaly situations occur; and (2) this advantage becomes stronger as the number of anomalies increases.

Furthermore, our previous research found that OCS (a) shows its effectiveness in other domains, (b) maintains its effectiveness for large-scale problems, and (c) achieves a better performance than conventional LCSs [23], even if each learning mechanism in OCS is simple and ordinary. From this fact, OCS currently has the capability of *robustness* in addition to *generality*, *scalability*, and *high performance*. Investigations of this type extend the applicable range of the OCS architecture in multiagent environments.

Future research will include: (1) investigations on the robustness of OCS for other examples; (2) analyses involving all types of anomalies; (3) an improvement on the robustness of OCS when starting from initial placement or when the number of anomalies is small; (4) a determination of both precise applicable ranges and limitations of OCS through a direct comparison with actual current systems and other architectures including ZCS and XCS; and (5) explorations into other capabilities of OCS.

References

1. **Argyris C, Schön DA** (1978) Organizational Learning, Addison-Wesley
2. **Arthur WB, Holland JH, Palmer R, Tayler P** (1997) Asset pricing under endogenous expectations in an artificial stock market, In: Arthur WB, Durlauf SN, Lane DA (Eds) The Economy as an Evolving Complex System II, Addison-Wesley, pp. 15–44
3. **Bull L, Fogarty TC, Snaith M** (1995) Evolution in multi-agent systems: evolving communicating classifier systems for gait in a quadrupedal robot, The 6th International Conference on Genetic Algorithms (ICGA '95), pp. 382–388
4. **Bull L** (1999) On evolving social systems: communication, special and symbiogenesis, Computational and Mathematical Organization Theory (CMOT), Kluwer Academic Publishers, Vol. 5, No. 3, pp. 281–301
5. **Carse B, Fogarty TC, Munro A** (1995) Adaptive distributed routing using evolutionary fuzzy control, The 6th International Conference on Genetic Algorithms (ICGA '95), pp. 389–397
6. **Cohen MD, Sproull LS** (1995) Organizational Learning, SAGE Publications
7. **Dorigo M, Schnepf U** (1992) Genetics-based machine learning and behavior-based robotics: a new synthesis, IEEE Transactions on System, Man, and Cybernetics **22**(6): 141–154
8. **Dorigo M, Colombetti M** (1998) Robot Shaping: An Experiment in Behavior Engineering, The MIT Press
9. **Gasser L, Bond A** (1988) Readings in Distributed Artificial Intelligence, Morgan Kaufman Publishers
10. **Goldberg DE** (1989) Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley
11. **Grefenstette JJ** (1988) Credit assignment in rule discovery systems based on genetic algorithms, Machine Learning, **3**: 225–245
12. **Holland JH** (1975) Adaptation in Natural and Artificial Systems, University of Michigan Press, Michigan
13. **Holland JH, Reitman J** (1978) Cognitive systems based on adaptive algorithms, In: Waterman DA, Hayes-Roth F (Eds) Pattern Directed Inference System, Academic Press
14. **Kim D** (1993) The link between individual and organizational learning, Sloan Management Review, Fall, pp. 37–50
15. **Lanzi PL, Stolzmann W, Wilson SW** (Eds) (2000a) Learning Classifier Systems. From Foundations to Applications, Springer-Verlag
16. **Lanzi PL, Wilson SW** (2000b) Toward optimal classifier system performance in non-Markov environments, Evolutionary Computation **8**(4): 393–418
17. **Potter M, Jong KD, Grefenstette J** (1995) A coevolutionary approach to learning sequential decision rules, The 6th International Conference on Genetic Algorithms (ICGA '95), pp. 366–372
18. **Russell SJ, Norving P** (1995) Artificial Intelligence: A Modern Approach, Prentice-Hall International
19. **Seredynski F, Cichosz P, Klebus G** (1995) Learning classifier systems in multi-agent environments, The First IEE/IEEE Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, pp. 287–292
20. **Smith SF** (1983) Flexible learning of problem solving heuristics through adaptive search, 1983 International Joint Conference on Artificial Intelligence (IJCAI '83), pp. 422–425
21. **Takadama K, Nakasuka S, Terano T** (1998) Multiagent reinforcement learning with organizational-learning oriented classifier system, IEEE 1998 International Conference On Evolutionary Computation (ICEC '98), pp. 63–68
22. **Takadama K, Terano T, Shimohara K, Hori K, Nakasuka S** (1999) Making organizational learning operational: implication from learning classifier system, Computational and Mathematical Organization Theory (CMOT), Kluwer Academic Publishers, Vol. 5, No. 3, pp. 229–252
23. **Takadama K, Terano T, Shimohara K, Hori K, Nakasuka S** (2001) Towards a multiagent design principle ∼ analyzing an organizational-learning oriented classifier system ∼, In: Loia V, Sessa S (Eds) Soft Computing Agents: New Trends for Designing Autonomous Systems, The Series of Studies in Fuzziness and Soft Computing, Springer-Verlag (to appear)
24. **Tomlinson A, Bull L** (1999) On corporate classifier systems: increasing the benefits of rule linkage, The 1999 Genetic and Evolutionary Computation Conference (GECCO '99), pp. 649–656
25. **Weiss G** (1999) Multiagent Systems – Modern Approach to Distributed Artificial Intelligence –, The MIT Press
26. **Wilson SW** (1994) ZCS: A zeroth level classifier system, Evolutionary Computation **2**(1): 1–18
27. **Wilson SW** (1995) Classifier fitness based on accuracy, Evolutionary Computation **3**(2): 149–175
28. **Wilson SW** (1998) Generalization in the XCS classifier system, The Third Annual Genetic Programming Conference (GP '98), pp. 665–674