XCS with an internal action table for non-Markov environments

Tomohiro Hayashida
Graduate School of Engineering,
Hiroshima University,
1-4-1, Kagamiyama, Higashi-Hiroshima,
Hiroshima, 739-8527, JAPAN
Email: hayashida@hiroshima-u.ac.jp

Ichiro Nishizaki
Graduate School of Engineering,
Hiroshima University,
1-4-1, Kagamiyama, Higashi-Hiroshima,
Hiroshima, 739-8527, JAPAN
Email: nisizaki@hiroshima-u.ac.jp

Keita Moriwake
Graduate School of Engineering,
Hiroshima University,
1-4-1, Kagamiyama, Higashi-Hiroshima,
Hiroshima, 739-8527, JAPAN

***Abstract*—To cope with sequential decision problems in non-Markov environments, learning classifier systems using the internal register have been proposed. Since, by utilizing the action part of classifiers, these systems control the internal register in the same way as choosing actions to the environment, they do not always work well. In this paper, we develop an effective learning classifier system with two different rule sets for internal and external actions. The first one is used for determining internal actions, that is, rules for controlling the internal register. It provides stable performance by separating control of the internal register from the action part of classifiers, and it is represented by "If [*external state*] & [*internal state*] then [*internal action*]," and we call a set of the first rules the internal action table. The second one is for selecting external actions as in the classical classifier system, but its structure is slightly different with the classical one; it is represented by "If [*external state*] & [*internal state*] & [*internal action*] then [*external action*]." In the proposed system, aliased states in the environment are identified by observing payoffs of a classifier and referring to the internal action table. To demonstrate the efficiency and effectiveness of the proposed system, we apply it to woods environments which are used in the related works, and compare the performance of it to those of the existing classifier systems.**

***Keywords*—Learning classifier systems; Non-Markov environments; XCS; Internal register.**

I. Introduction

Although classifier systems with if-then rules which develop through interaction with environments were initially considered as a computational model for cognition [12], [14], they are now widely applied to many areas, including autonomous robotics [8], [29], classification and data mining [33], [25], [15], traffic signal control [2], [4], and FPGA design [6].

A framework of classifier systems was initially proposed by Holland [11], [12], and subsequently a wide variety of classifier systems have been developed [7], [31], [32]. Especially, XCS developed by Wilson [32] has been attracting a lot of attention, and it is publicly recognized as one of the most successful learning classifier systems. Before XCS, the fitness of a classifier was calculated by using the expected payoff or the strength in the traditional learning classifier systems, and therefore there was a problem that classifiers which have low expected payoffs but are required to find optimal policies are eliminated by the procedure of genetic algorithms. To overcome this difficulty, the degree of accuracy is used as the fitness in XCS, and it is based on the difference between the predicted payoff and the actually received payoff.

In this paper we deal with non-Markov environments or partially observable Markov decision processes. In Markov environments where the probability of being in a given state depends on the current state and action but not on any past states or actions, agents can select the optimal policy by appropriately utilizing the information of the environment. If even in a Markov environment an agent can obtain only restrictive information of the environment, such a process is called a partially observable Markov decision process (POMDP). In a POMDP, different states can exist even if agents obtain the same information from the environment, and then the agents are said to suffer from a perceptual aliasing problem. In an aliased position or state, an agent cannot identify the current situation only through the information obtained from the environment by itself, and then it cannot select the next optimal action. From this reason, one can understand that the learning method of an agent in non-Markov environments is similar to that of POMDPs.

Since XCS determines an action by using the information about the environment at the current period, it is difficult to select an appropriate action in a non-Markov environment involving aliased states which cannot be discriminated only by the information about the environment at the current period. Several attempts using reinforcement learning and learning classifier systems for finding optimal policies in a non-Markov environment or a POMDP have been reported. For instance, Pineau et al. [22] propose an algorithm based on reinforcement learning for POMDPs, and apply it to a robot domain problem where an agent searches for and tags a moving opponent. Roy et al. [24] try to solve large scale POMDPs problems by reducing the dimensionality of the problem space. Shani et al. [26] present a learning model for POMDP based on reinforcement learning with memories of tree structure. Methods based on classifier systems such as ZCS [5] and ACS [27] have been also developed and applied to the grid-like woods environments which are benchmark problems for POMDPs. Moreover, Lanzi and Wilson [20] develop XCSM and XCSMH which are extensions of XCS, and intends to resolve environmental aliasing by incorporating the internal registers. In XCSM, both an external action which means an action that the agent takes in the environment and an internal action for controlling the internal register are specified in the

action part of a classifier, and they are treated in the same way. Although implementation for using the internal register is simple and elegant, its performance is not always good as we will show the experimental result. Moreover, it is difficult to determine an appropriate size of the internal register, and if it is too large for a given problem, the space of exploration becomes larger than necessary. Recently, Hamzeh et al. [10] develop the parallel specialized XCS (PSXCS), Zang et al. [35] develop XCS with average reward (XCSAR) which the Q-learning employed by XCS is replaced to R-learning not to limit the length of action chains. Preen and Bull [23] introduce discrete and fuzzy dynamical system within XCSF learning classifier system [34].

In PSXCS, along the lines of the history window approach [16] the information of the environments and the selected actions are recorded and aliases states are identified by the condition part of classifiers corresponding to the history of the environments and the selected actions.

Reinforcement learning is a type of machine learning such that an agent selects an action in an environment so as to maximize the cumulated sum of reward function. The agent receives the reward from the environment after taking an action, and by repeating this procedure it learns to take an appropriate policy so as to maximize the reward. In non-Markov environments or POMDPs, the agent cannot always obtain the optimal policy through the usual implementation of reinforcement learning. By using some ideas such as referring to the history of actions which are taken by the agent and the perceived information about the environment or reducing the dimensionality of the problem space, systems of reinforcement learning are improved [9], [22], [24], [26], [30]. Since reinforcement learning acquires exhaustive rules for selecting appropriate actions to an intended problem and then it holds a sufficient number of rules to deal with possible states of the problem, it works efficiently for relatively small-scale problems. However, for large scale problems or problems with many aliased states, it may perform poorly because of explosive growth in the number of rules and the use of memories.

In classifier systems the idea of reinforcement learning is implemented in a sense that Q-learning-like payoff is computed, and classifier systems are extended so as to cope with non-Markov environments or POMDPs [1], [10], [17], [18], [20], [28]. An agent in a classifier system holds rules in if-then type called classifiers, and it employs an action specified in a classifier such that the condition of the classifier matches the information from the environment. In particular, *don't care* denoted by # is introduced in the condition part of classifiers, and conditions corresponding to # match all states. By this capability the rules represented by classifiers are generalized, that is, the agent acquires the ability to hold classifiers matching multiple different states of the environment. Compared to reinforcement learning, it is thought that the number of rules is smaller and memories are efficiently used in classifier systems, and genetic algorithms can be applied to a set of rules represented in if-then format without difficulty for evolving the rule set suitably. From these features of classifier systems, it is adequate to apply them to problems in non-Markov environments or POMDPs.

In this paper, we develop a learning classifier system for non-Markov environments or POMDPs where a mechanism for controlling the internal register is separated from classifiers and aliased states are identified by detecting fluctuation of the payoffs received by classifiers. We call the proposed system XCSAT (XCS with an internal Action Table) because it is characterized by an internal action table which is a set of rules for identifying aliased states. In XCSAT, after detecting the fluctuation of payoffs which means the existence of aliased states, the environmental information and the corresponding update of the internal register are recorded in the internal action table as a rule for updating the internal register. By controlling the internal register through the information from the internal action table, more efficient and stable performance can be expected in XCSAT.

The remainder of this paper is organized as follows. After describing non-Markov environments in section 2, we mention the properties of XCSM and XCSMH developed by Lanzi and Wilson [20] in section 3. In section 4, we propose a learning classifier system with the internal action table, XCSAT, in which aliased states are identified by detecting fluctuation of payoffs and referring to the internal action table. The experimental result of XCSAT is shown, compared with XCSM and XCSMH in section 5, and finally, section 6 concludes with some comments.

II. Non-Markov Environments

Markov environments have memoryless property, that is, in Markov environments the probability of being in a given state depends on the current state and action but not on any past states or actions, and environments without such property are said to be non-Markov environments. Learning classifier systems for non-Markov environments have been proposed, and to evaluate their performances, woods environments which are grid-like non-Markov environments are used [1], [17], [18], [20], [28].

First of all, to understand that it is difficult for learning classifier systems which are not developed specially for non-Markov environments to find an optimal policy in non-Markov environments, we illustrate actions of an agent in a simple woods environment termed **Woods100** [18], which is shown in Fig. 1.

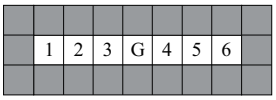

Fig. 1: Woods environment: **Woods100**

In **Woods100**, there are 7 cells which are cell 1 to cell 6 and cell G meaning the goal, and the 7 cells are surrounded by the walls. Although the agent can generally move to 8 possible directions (N, S, E, W, NE, SE, NW, and SW) in a woods environment. In **Woods100**, the agent in any of the 7 cells can move only to W (left) or E (right). The agent starts from cell 1 or cell 6, and it tries to reach cell G. Since the agent moves either to left or to right, the condition part of classifiers deals with states of cells located on only both sides of the agent.

In Table I, classifiers which lead the agent in each cell to the goal are enumerated, and "w" indicates the wall, "c"

TABLE I: Classifiers corresponding to **Woods100** and cells

condition		action	cell
left	right	direction	
w	c	right	1
c	w	left	6
c	c	right	2, 5
c	c	left	2, 5
c	G	right	3
G	c	left	4

(w: wall, c: corridor, G: goal)

indicates the corridor, and "G" indicates the goal in the first and second columns. Take a classifier in the first row of Table I as an example. The first classifier means "If [{*left*: w} and {*right*: c}] then [*external action*: move right]." Therefore, one finds that this classifier should be selected in cell 1, which is given in the rightmost column of Table I.

Since there are 6 available cells except for the goal cell, an optimal policy can be described by 6 classifiers as shown in Table I, and there are two classifiers with the condition parts matching both of the environmental states corresponding to cells 2 and 5. Although the environmental information perceived by the agent in cell 2 is the same as that in cell 5, optimal actions in the two cells are different. From this fact, these two cells are aliased states for the agent, and the agent informed of only the environmental information at the current period cannot find the optimal policy. Thus, it follows that a woods environment such as **Woods100** is one of non-Markov environments.

III. Learning Classifier Systems with Internal Memory

To cope with environmental aliasing, Lanzi and Wilson [20] develop XCSM (XCS with internal memory) which is an extension of XCS. In XCSM, a condition for the internal register and an action for controlling the internal register are added to the condition part and the action part of a classifier, respectively.

TABLE II: Classifiers of XCSM and the related information

no.	condition part			action part		cell	payoff
	left	right	register	direction	revised register		
a	w	c	0	right	0	1	$\gamma^2 R$
b	c	c	0	right	0	2, 5	γR
c	c	G	0	right	0	3	R
d	c	w	0	left	1	6	$\gamma^2 R$
e	c	c	1	left	1	2, 5	γR
f	G	c	1	left	1	4	R
g	w	c	1	right	0	1	$\gamma^2 R$
h	c	G	1	right	0	3	R
i	c	w	1	left	1	6	$\gamma^2 R$
j	G	c	0	left	1	4	R

In Table II, we give an example of an optimal policy in XCSM to **Woods100**, which can be obtained after enough learning process. Since it is necessary for XCSM to discriminate the two aliased states in **Woods100**, only the size of two is required for the internal register. Let the initial value of the internal register be 0. To utilize the internal register, the value of the internal register and its new value to be updated are added in the condition part and the action part of classifiers, respectively, as seen in Table II. For example, in classifier *a* given in Table II, a condition on the internal register "if [*internal register*: 0] (if the internal register is 0)" is given in addition to a condition on the environmental information "if [{*left*: w} and {*right*: c}] (if the left-side cell is the wall and the right-side cell is the corridor)," and an action to the internal register "[*internal action*: set 0] (set 0 in the internal register)" is also given in addition to an action to the environment "[*external action*: move right]." Although the information from the environment when being in cell 2 is the same as that in cell 5 in XCS, since in XSCM the value of the internal register is changed from 0 to 1 by classifier *d* used at cell 6 which is located on the right side of cell 5 but it is not changed to cell 2, cells 2 and 5 can be distinguished. It should be noted that a set of classifiers shown in Table II is an optimal policy, but there exist other sets of optimal policies such as an optimal policy of the reversed procedure.

A classifier in XCSM has the same parameter set as those of XCS: the prediction p, the prediction error ϵ, and the fitness F. The prediction p is a payoff that the system expects if the condition of the classifier conforms with the environmental state and the action of the classifier is performed. The prediction error ϵ estimates an error of the prediction p by using the Q-learning-like payoff. The fitness F means the accuracy of the prediction p and it is a function of the prediction error ϵ. Moreover, the learning process of XCSM is similar to that of XCS, and it is slightly modified for introducing the internal register.

Although XCSM can find an optimal policy as seen in Table II, depending on environments, the sequence of actions may not converge because actions to the environment and to the internal register are determined according to received rewards. We illustrate this difficulty by using **Woods100**. Let R denote the reward from the environment when the agent reaches the goal, and any reward is not paid by arriving at the other cells. When the sequence of actions converges, the payoffs received by the classifiers are shown in the rightmost column of Table 2, where γ is a discount factor.

Assume that the following classifier *a'* is included in the system in addition to the set of classifiers given in Table II:

(*a'*) If [{*left*: w} and {*right*: c}] & [*internal register*: 0] then [*external action*: move right] & [*internal action*: set 1].

Classifier *a'* is the same as classifier *a* except for the internal action, which means the value of the internal register to be updated, in the action part of classifiers. Although, as a matter of course, using classifier *a'* instead of classifier *a* is not optimal, the payoff of classifier *a'* is $\gamma^2 R$ which is the same as that of classifier *a* if classifiers *e* and *b'* are used, where *b'* is the same as classifier *b* except for the internal action. Thus, since the fitness F is a function of the payoff, it is possible that classifier *a'* is substituted for classifier *a*, and therefore it is difficult to generate an optimal policy stably. Beside, it should be noted that when the size of the internal register becomes larger, the performance of XCSM grows worse due to increase of the search space.

To improve the performance of XCSM, XCSMH is also

proposed as an extension of XCSM, and the following modifications are remarked.

(i) The value of the internal register is changed only if the environmental information perceived by the agent changes as a result of the executed action to the environment.
(ii) The actions to the environment and to the internal register are performed in a stepwise fashion. After the value of the internal register is determined by the greedy method, the action to the environment is selected by the ϵ-greedy method.

For example, by reason of (i), it is not possible that the direction to move to a cell of wall is chosen and the value of the internal register is updated at the same time. The modification of (ii) facilitates the combination of actions to the environment and treatment of the internal register, and then it is thought that the performance is improved.

However, XCSMH does not resolve the above mentioned difficulty essentially, and we need some solution to effectively manage the internal register. In this paper, focusing on fluctuation of payoffs of classifiers used in aliased states, we propose an effective learning classifier system with an internal action table providing stable performances by separating the control of the internal register from the action part of classifiers.

IV. Classifier system with an internal action table

As we pointed out before, in non-Markov environments or POMDPs, although XCSM can find an optimal policy, depending on environments, its performance is not always stable because actions to the environment and to the internal register are determined according to received rewards. We will show this fact by some computational experiments in the following section. In this paper, we develop a learning classifier system called XCSAT (XCS with an internal Action Table) for non-Markov environments or POMDPs where controlling the internal registers is separated from classifiers and aliased positions or states are identified by detecting the fluctuation of the payoffs received by classifiers. In XCSAT, after detecting the fluctuation of payoffs, the corresponding environmental information and the updated value of the internal register are recorded into the internal action table as a rule for updating the internal register. By introducing the above mentioned two features simultaneously, XCSAT works efficiently for non-Markov environments or POMDPs.

To check whether or not a position that the agent have arrived is an aliased one, XCSAT focuses on the fluctuation of payoffs received by classifiers. The maximum and the minimum payoffs are recorded together with the corresponding periods of time. If the difference between the maximum and the minimum is larger than the threshold after a given amount of periods had elapsed, XCSAT judges that the payoffs of the classifier fluctuate.

If the payoffs of the classifier executed at the present moment, say period t, does not fluctuate and the payoff fluctuation is observed at period $t-1$, XCSAT judges that the environment at period $t-1$ is an aliased state. To utilize the information about such aliased states, the system records the external state, the internal state and the internal action which are observed and selected at period $t-2$ into the internal action table. By referring to the internal action table with the information about the aliased states each period, XCSAT can identify each aliased state and select an appropriate action for the aliased state.

A. Rule representation and the internal action table

In the proposed method, states of the environment are identified by observing the payoffs received by classifies and referring to the internal action table. To do so, the system stores rules for updating the internal register in the internal action table. Unlike XCSM and XSCMH, XCSAT does not use classifiers to control the internal register, but to this end it uses the internal action table in which the history of use of the internal register is stored.

To describe the learning procedure of XCSAT, we define the following technical terms. Let "an *external state*" be an environmental state, "an *internal state*" be the value of the internal register, "an *external action*" be an action taken by the agent to the environment, and "an *internal action*" be the value of the internal register to be updated.

Using these terms, we represent a classifier in XCSM or XCSMH by the following if-then rule:

If [*external state*] & [*internal state*]
then [*external action*] & [*internal action*].

It should be noted that an *internal action* is specified in the action part of a classifier in XCSM or XCSMH. In contrast, a classifier in XCSAT is expressed as

If [*external state*] & [*internal state*] & [*internal action*]
then [*external action*],

where, in the action part, there does not exist an *internal action*, but it is in the condition part. The *internal action* in the condition part is utilized to update the parameters of a classifier when the classifier is selected to activate to the environment. Apart from classifiers, rules for updating the internal register are stored in the internal action table in the following form:

If [*external state*] & [*internal state*] then [*internal action*].

If the environmental state and the value of the internal register coincide with the values of the *external state* and *internal state* of a rule in the internal action table, respectively, the value of the internal register is updated by using the value of the *internal action* of the rule in the internal action table for updating the internal register. Since the value of the internal register is determined as just described, classifiers in XCSAT have no information about *internal actions* in the action part.

B. Update and usage of the internal action table

Using **Woods100** shown in Fig. 1 and Table II, we illustrate the fluctuation of payoffs received by classifiers used in aliased states. Assume that the sequence of actions of the agent converges through enough learning process. A payoff of classifier *c* which is used at cell 3 and leads to the goal, cell *G*, and that of classifier *f* which is used at cell 4 and also leads to

cell *G* are the same value R. If classifier *b* is instantly used at cell 2 and then classifier *c* is used at cell 3, classifier *b* receives the payoff of γR, where γ is a discount factor. However, if, after classifier *e*, which should be used ideally at cell 5, is used at cell 2, the agent returns to cell 2, classifier *b* is used at cell 2 and then finally classifier *c* is used at cell 3, then classifier *b* receives the payoff of $\gamma^3 R$. If classifier *e* is used at cell 2 repeatedly, the payoff of classifier *b* becomes smaller. Thus, the payoff of classifier *b* ranges from $\gamma^3 R$ to some small value, and as for classifier *e*, a similar fluctuation of the payoff can be observed. Moreover, since the payoffs of classifiers *a* and *d*, which should be used ideally at cells 1 and 6, respectively, are calculated from those of classifiers *b* and *e*, they also fluctuate. From this observation, if the payoff of a classifier used at a certain cell, say cell *x*, fluctuates and cell *x* is adjacent to a cell such that the payoff of the corresponding classifier does not fluctuate, it can be inferred that an environmental state when being in cell *x* is an aliased state. To utilize such information, rules for identifying aliased states are stored in the internal action table.

Although, in XCSAT, an external action which is an action taken by the agent to the environment is selected among classifiers matching the environmental state, an internal action for updating the internal register is determined by finding a rule conforming with the external state and the internal state in the internal action table. As we mentioned above, the form of rules in the internal action table is "If [*external state*: $\cdots$] & [*internal state*: $\cdots$] then [*internal action*: $\cdots$]," and a rule conforming with the external state and the internal state perceived by the agent is searched in the internal action table. An internal action of the rule selected from the internal action table is performed. By doing so, XCSAT can identify aliased states and select appropriate external actions. Eventually, the fluctuation of classifiers' payoffs disappears and an optimal policy can be found. If the payoff fluctuation of classifiers is still observed, it follows that there exist aliased states which are not identified by the system yet.

We demonstrate a process of updating the internal register by using **Woods 100** shown in Fig. 1. Examples of classifiers and the internal action table of XCSAT are given in Tables III and IV. In the course of repetition of trials in **Woods100**, suppose that the fluctuation of payoffs received by a classifier is observed and it is revealed that an environmental state when being in cell 2 is an aliased state. At this point, a rule for the internal register "If [{*left*: w} and {*right*: c}] & [*internal register*: 0] then [*internal action*: set 1]" is recorded in the internal action table, and this rule for the internal register corresponds to cell 1. Moreover, at the same time, a new classifier corresponding to the same external state and the internal state that the value of the internal register is 1 ([*internal register*: 1]) is added into the system. More specifically, the following classifier is generated: If [{*left*: w} and {*right*: c}] & [*internal register*: 1] & [*internal action*: set 1] then [*external action*: move right].

In general, if XCSAT finds an aliased position or state, a rule for identifying the aliased state is recorded in the internal action table. By referring to the internal action table, XCSAT can efficiently distinguish positions of the agent. More precisely, if the payoffs received by the classifier executed at period t does not fluctuate and the payoff fluctuation at period

TABLE III: Classifiers of XCSAT for woods100 and the related information

no.	condition part: external state left	condition part: external state right	condition part: internal register	condition part: internal action	action part: external action	cell	payoff
a	w	c	#	#	right	1	$\gamma^2 R$
b	c	c	1	1	right	2, 5	γR
c	c	G	#	#	right	3	R
d	c	w	#	#	left	6	$\gamma^2 R$
e	c	c	0	0	left	2, 5	γR
f	G	c	#	#	left	4	R

TABLE IV: Internal action table of XCSAT for woods100

no.	condition part: external state left	condition part: external state right	condition part: internal register	action part: internal action	cell
a	w	c	0	1	1
b	c	w	1	0	6

$t-1$ is observed, it is judged that the environment at period $t-1$ is an aliased state. To execute this procedure successfully, XCSAT records the external state, the internal state and the internal action which are observed and selected at period $t-2$ in the internal action table. Thereafter, by referring to the internal action table, it acquires ability to distinguish such states of the environment. The data insertion of the internal action table and the generation of the corresponding classifier are summarized as follows.

Step 1 Refer to the internal action table, and then if XCSAT finds a rule with the condition matching the current environment and the internal register, update the internal register to the value specified by the rule.

Step 2 Execute an action specified by a selected classifier.

Step 3 If the payoff fluctuation is observed, set the flag for update on and return to Step 1. Otherwise, go to Step 4.

Step 4 If the flag is on, go to Step 5. Otherwise, return to Step 1.

Step 5 Add the information of the external state, the internal state and the internal action which are observed and selected at the period before last in the form of

If [*external state*: $\cdots$] & [*internal state*: $\cdots$] then [*internal action*: $\cdots$],

into the internal action table. Moreover, add a new classifier consisting of the information from the environment, the value of the internal register, the updated value of the internal register and the executed external action at the last period in the form of

If [*external state*: $\cdots$] & [*internal state*: $\cdots$] & [*internal action*: $\cdots$] then [*external action*: $\cdots$].

Then, after setting the flag off, return to Step 1.

Some explanatory remarks on this procedure follows.

- After the elapse od the given periods, say 1000 periods, XCSAT starts to refer the internal action table because it needs enough learning time for external environments except aliased states.

- In Step 3, if the difference between the maximum and the minimum of the payoffs received by the selected classifier is larger than the threshold, XCSAT concludes that the payoff fluctuation of the classifier is observed.
- The rules for the internal register are not deleted unless the number of rules exceeds the capacity for the internal action table, and if it exceeds the capacity, the rule with the lowest use is replaced with a new rule for the internal register.
- If two or more aliased states adjoin and there exist multiple such adjoining aliased states, the fluctuation of the payoffs could not be always suppressed. When the payoff fluctuation cannot be suppressed within a given amount of periods after the last update of the internal action table, even if the condition of Step 4 is not satisfied, with a given probability a new rule for the internal register is added into the internal action table.

C. Algorithm of XCSAT

The algorithm of XCSAT is summarized as follows.

Step 1 After perceiving the current external and internal states, XCSAT finds all the classifiers satisfying these conditions. A set of these classifiers are called the match set $[M]$.

Step 2* If a rule matching the perceived external and internal states is found in the internal action table, an internal action specified by the rule is executed. Otherwise, reset the internal register, i.e., set 0 at the internal register.

Step 3 For classifiers with the executed internal action in $[M]$, a prediction array is calculated by using the prediction and the fitness. From the prediction array, an external action is determined by the greedy or the ϵ-greedy method. A set of classifiers with the selected external action in $[M]$ is called the action set $[A]$.

Step 4 After updating the parameters of each classifier in $[A]$, the genetic operations of reproduction, crossover and mutation are performed to the condition part of the classifiers.

Step 5* If the condition based on the payoff fluctuation for updating the internal action table is satisfied, the corresponding external state, internal state, and internal action are recorded in the internal action table.

Step 6 If the agent reaches the terminal position, the algorithm stops. Otherwise, go to Step 1,

It should be noted that as mentioned in the previous subsection, to find appropriate actions for non-aliased states, for the given initial ceratin periods, XCSAT does not refer the internal action table, and therefore Steps 2 and 5 marked with an asterisk, which involve reference and update to the internal action table, are skipped in the initial certain periods, namely it performs the same procedure as that of XCS.

Let the ith classifier be denoted by cl_i. Similarly to those of XCS [3], [20], the main parameters of classifier cl_i are the prediction $cl_i.p$, the prediction error $cl_i.\epsilon$, and the fitness $cl_i.F$. These parameters are updated based on the payoff P received by a classifier and the other parameters. A classifier in the action set $[A]$ receives the following Q-learning-like payoff:

$$P = \begin{cases} R, & \text{when reaching the termination position} \\ P_{-1} + \gamma \max PA, & \text{otherwise,} \end{cases} \tag{1}$$

where R is the reward from the environment, P_{-1} is the payoff at the previous period, PA is the prediction array at the current period, and γ is a discount factor. For a given external action a_i, an element of PA is calculated as follows:

$$PA(a_i) = \sum_{cl_k \in [M]_{\hat{m},a_i}} cl_k.p \cdot cl_k.F \Big/ \sum_{cl_k \in [M]_{\hat{m},a_i}} cl_k.F, \tag{2}$$

where $[M]_{\hat{m},a_i}$ is a set of classifiers in $[M]$ such that an *internal action* is the executed internal action $\hat{m}$ and an *external action* is a_i. In Step 3, by using the prediction array PA, an external action is determined.

The prediction $cl_i.p$ and the prediction error $cl_i.\epsilon$ are updated as follows:

$$cl_i.p = cl_i.p + \beta(P - cl_i.p), \tag{3}$$

$$cl_i.\epsilon = cl_i.\epsilon + \beta(|P - cl_i.p| - cl_i.\epsilon), \tag{4}$$

where β is the learning rate. The smaller the prediction error $cl_i.\epsilon$, the larger the fitness $cl_i.F$ becomes. To this end, the accuracy $cl_i.\kappa$ is defined as

$$cl_i.\kappa = \begin{cases} 1 & \text{if } cl_i.\epsilon < \epsilon_0 \\ \alpha \left(\frac{cl_i.\epsilon}{\epsilon_0} \right)^{-\nu} & \text{otherwise,} \end{cases} \tag{5}$$

where α, ν, and ϵ_0 are parameters, the fitness $cl_i.F$ is calculated as follows:

$$cl_i.F = cl_i.F + \beta(cl_i.\kappa' - cl_i.F), \tag{6}$$

where $cl_i.\kappa' = cl_i.\kappa / \sum_{cl_k \in [A]} cl_k.\kappa$.

As for the genetic operations described in Step 4, if the average elapsed time periods of classifiers in the action set $[A]$ after the last genetic operations for them is larger than a given time period θ_{GA} in advance, the genetic algorithm are executed to the parts of classifiers describing the external conditions. Overgeneral rules in XCSAT are removed in the same way as in XCS. That is, since the prediction errors of overgeneralized classifiers become large and then their fitness in the genetic algorithm described in Step 4 degrades, such classifiers are not reproduced eventually. Two classifiers are chosen by using the roulette wheel selection, and the one-point crossover is applied to them. If a gene chosen for mutation is #, which means "don't care," the perceived external state is filled in the gene. Otherwise, it is exchanged for #.

To judge the fluctuation of the payoffs in Step 5, the maximal payoff $cl_i.p_{\max}$ and the minimal payoff $cl_i.p_{\min}$ are recorded together with the corresponding periods $cl_i.t_{\max}$ and $cl_i.t_{\min}$. Let P be the payoff of classifier cl_i. If $P > cl_i.p_{\max}$, the maximal payoff $cl_i.p_{\max}$ is updated, and similarly if $P < cl_i.p_{\min}$, the minimal payoff $cl_i.p_{\min}$ is updated. Let $cl_i.exp$ be the number of updating, and θ_p and θ_t be parameters. If $cl_i.p_{\max} - cl_i.p_{\min} < \theta_p$ and $cl_i.exp > \theta_t$, XCSAT judges that the payoff of classifier cl_i does not fluctuate. Otherwise, it

judges that the payoff of the classifier fluctuates. Furthermore, let θ_r be a parameter. If the elapsed time periods after the last update of either $cl_i.p_{\max}$ or $cl_i.p_{\min}$ is larger than θ_r, the not yet updated parameter and the update counter $cl_i.exp$ are initialized.

V. Computational experiment

To demonstrate the effectiveness of XCSAT, we perform a computational experiment by using a woods environment **Woods101$\frac{1}{2}$**, and compare XCSAT with XCSM and XCSMH. Furthermore, with another eight woods environments [1], [17], [20], [21], [28], we examine the performance of XCSAT.

A. *Woods101$\frac{1}{2}$*

In XCSAT, the agent perceives substances of the adjacent eight cells (N, S, E, W, NE, SE, NW, and SW), and it distinguishes among "wall," "corridor," and "the goal" of substance of each of the cells. As seen in Fig. 2, **Woods101$\frac{1}{2}$** is a separated symmetric woods environment, and the agent tries to move from any cell labeled as S to one of the cells labeled as G in the shortest possible route.

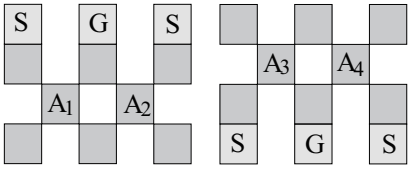

Fig. 2: Woods environment: **Woods101$\frac{1}{2}$**

Since, in the cells labeled as A_1, A_2, A_3, and A_4, substances of the eight cells that the agent perceives are the same, the agent cannot distinguish these states of the environment. In cells A_1, A_2, A_3, and A_4, "move upper right," "move upper left," "move lower right," and "move lower left" are appropriate actions, respectively.

An episode is defined as a process from starting at cell S to reaching cell G. Let one trial be 8000 episodes; the periods until episode 6000 are served for exploring or learning, and the remaining 2000 episodes are used for test of the performance. While the ϵ-greedy method which includes stochastic selection of actions is employed in the learning periods, the greedy method in which an action with the largest prediction is chosen with certainty is used in the test periods. To examine the performances of XSCAT, XCSM and XCSMH, data from the last 1000 episodes are used for each trial, and their performances are evaluated by the average of 30 trials. The parameters used in the computational experiment are shown in Table V. In the experiment, we use XCSM and XCSMH programs of our own making according to the procedure given in Lanzi and Wilson [20]. The sizes of the internal registers in three programs, XCSAT, XCSM and XCSMH, are all 4 for the seven problems in sections 5.1 and 5.2, and they are 6 and 8 for the two problems, **Lab1** and **LargeMaze**, in section 5.3., respectively.

In Fig. 3 and Fig. 4, we compare the performances of XCSAT, XCSM, and XCSMH, varying the exploration rate ϵ in the ϵ-greedy method in the learning periods. The rate of convergence in Fig. 3 is the percentage of success in the 30 trials, and the success means that the agent exactly takes a shortest route and reaches the goal in the last 1000 episodes in the test periods. Aside from this, the rate of convergence in Fig. 4 is the percentage in the 30 trials that the agent takes the same fixed route including the shortest route in the last 1000 episodes. Therefore, the term "the convergence" means that the agent takes the same route for a given starting point in the last 1000 episodes.

TABLE V: Parameters

learning rate	$\beta = 0.2$	discount factor	$\gamma = 0.71$
periods for GA	$\theta_{GA} = 25$	crossover probability	$p_c = 0.75$
mutation probability	$p_m = 0.025$	accuracy parameters	$\alpha = 0.1$, $\nu = 5$
payoff range	$\theta_p = 5$	updating counter	$\theta_t = 30$
periods for updating	$\theta_r = 10$		

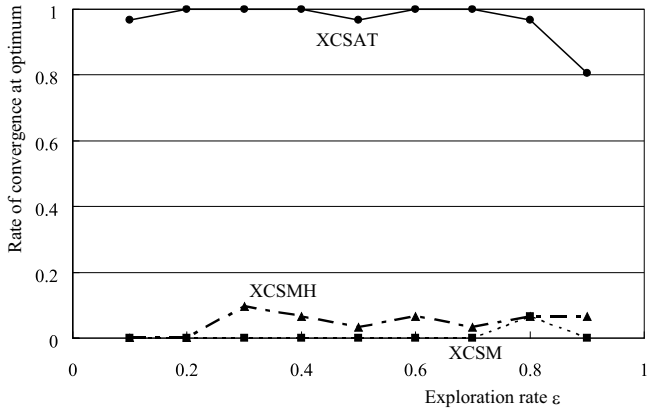

Fig. 3: Convergence on the shortest routes

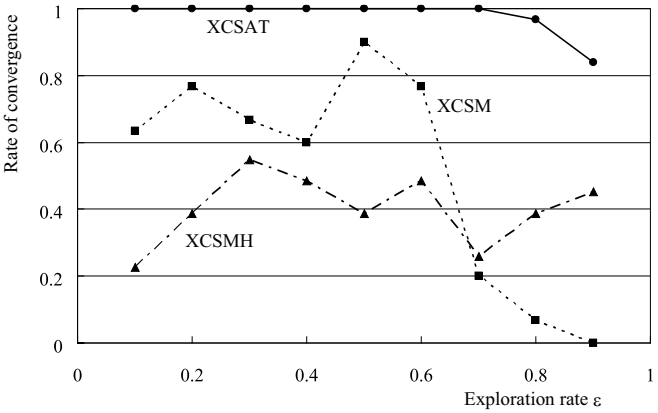

Fig. 4: Convergence on fixed routes

As seen in Fig. 3 and Fig. 4, while XCSM and XCSMH can hardly find the shortest routes, the agent in XCSAT pursues the shortest routes with great accuracy. Although the rate of convergence on the shortest routes in XCSMH is slightly larger than that of XCSM, for the convergence on another fixed route, the rate of XCSM is larger than that of XCSMH. The number

of steps taken from the starts to the goals in XCSM and XCSMH is larger than 12 on an average, and thus it follows that the routes taken by the agent in these systems converge some fixed routes including unnecessary actions because the number of steps for the shortest routes is 4.

B. Performance verification

We continue to examine the performance of XCSAT by using other 6 woods environments which are depicted in Fig. 6 in the appendix, and the summary data of them are given in Table VI [17], [20], [21], [28].

TABLE VI: Woods environments for the computational experiment

woods environment	number of all states	number of aliased states
woods101	11	4
woods102	28	10
maze7	10	2
mazeF4	11	2
maze10	19	13
Littman57	15	8

In these woods environments, starting cells are randomly chosen from among non-goal cells. We evaluate the performances by measuring the number of steps taken from the starts to the goals. In the experiment, if the number of actions taken by the agent is larger than 10000 and the agent still does not reach the goal, the current episode terminates and the next episode begins with a new starting cell. The exploration rate is fixed at $\epsilon = 0.5$. The other experimental conditions and the parameters are the same as those in the computational experiment for **Woods101$\frac{1}{2}$** given in section V-A.

The result of the computational experiment is given in Fig. 5 and Table VII. The performances of the three systems XCSAT, XCSM and XSCMH are compared on the basis of the data of the last 1000 episodes for the 30 trials. The term *best* in Fig. 5 and Table VII means the minimum among the results of the 30 trials where the result of each trial is the average of the last 1000 episodes. Therefore, we note that the *best* is not always the optimum. The terms *mean* and *worst* also mean the average of the 30 trials and the maximum among the 30 trials, respectively.

In Fig. 5, *best*, *mean*, *worst*, and the range of the steps taken by the agent from the starts to the goals are given graphically. In particular, *mean* is denoted by a circle, *best* and *worst* are denoted by bars, and the range of the steps is represented by vertical lines. In Table VII, the minimal steps among the three systems are emphasized by boldface, and for reference the average steps of the shortest routes are given in the rightmost column. For example, the average of shortest steps of *mazeF4* is calculated by summing up the numbers of the shortest steps to the goal for all cells and dividing the number of cells, i.e.,

$$(4+3+2+1+0+4+5+5+6+7+6)/11 = 3.90.$$

As seen in Table VII, the number of steps of XCSAT for each of the six woods environments is close to the average step of the shortest routes. The *best* of XCSMH is smaller than that of XCSM, and XCSMH provides comparable result to that of XCSAT. However, the *mean* and *worst* of XCSMH

TABLE VII: Results of the computational experiment (steps)

woods environment		XCSAT	XCSMH	XCSM	average of shortest steps
woods101	mean	**2.70**	22.38	3.19	2.45
	best	**2.62**	2.64	2.64	
	worst	**2.85**	295.92	4.11	
woods102	mean	**3.28**	6.29	6.73	2.57
	best	**3.00**	4.23	4.93	
	worst	**3.73**	14.63	12.45	
maze7	mean	**4.19**	44.51	38.57	3.70
	best	**3.90**	**3.90**	4.93	
	worst	**7.75**	392.63	126.26	
mazeF4	mean	**4.40**	116.89	35.72	3.90
	best	**4.09**	4.11	4.18	
	worst	**5.54**	1171.19	133.47	
maze10	mean	**6.51**	12.08	46.45	4.32
	best	**5.70**	6.39	8.20	
	worst	**8.54**	61.76	173.39	
Littman57	mean	4.19	**3.89**	6.85	3.47
	best	**3.47**	**3.47**	5.52	
	worst	5.99	**5.14**	9.87	

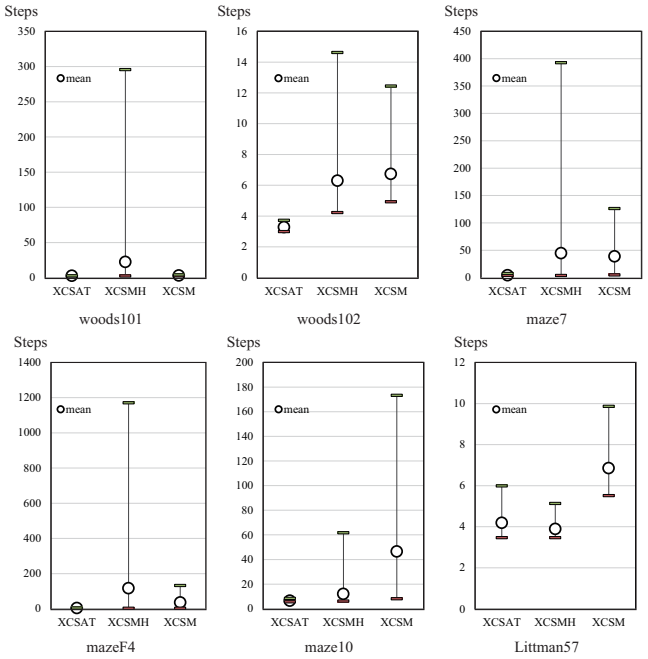

Fig. 5: Results of the computational experiment (steps)

are obviously larger than those of XCSAT, and in some woods environments the *mean* and *worst* of XCSMH sometimes are larger than those of XCSM. This means that the learning performance of XCSMH is not stable, and we consider that this difficulty is attributed to the problem described in section III. In contrast, XCSAT works well in finding the shorter routes to the goal, and then the performance of XCSAT is stable as seen Table VII. As we discussed in section V-A, also from the viewpoint of the convergence given in Fig. 3 and 4, the performance of XCSAT is more stable than those of XCSM and XCSMH. In general, the performance of XCSAT is superior to XCSM and XCSMH except for **Littman57**. In the experiment for **Littman57**, XCSMH shows the best performance but the performance of XCSAT is also good. Both of XCSAT and XCSMH find the shortest routes, and the

difference between the mean steps of them is only 0.3 steps.

As for Littman57, which is composed of 15 states including 8 aliased states, XCSMH works slightly better than XCSAT, and the performance of XCSM is also reasonable. For maze7 or mazeF4, however, XCSMH and XCSM operate inefficiently despite the fact that maze7 or mazeF4 is composed of 10 or 11 states and there are only two aliased states in both of them. This performance is thought to be due to the property of the optimal actions. That is, optimal actions are the same in the aliased states of Littman57, while they are different actions in those of maze7 or mazeF4. XCSAT works well in both problems because it finds an appropriate action efficiently by referring the internal action table.

C. Performance and adaptability for larger problems

The average numbers of all states and aliased states of the woods environments dealt with in section V-B are 15.7 and 6.5, respectively. To examine the effectiveness of XCSAT to lager problems, we use a woods environment **Lab1** [1] which is 5 times as large as the woods environments in section V-B, and we also provide a new woods environment **LargeMaze** which is 10 times as large as them. These woods environments are depicted in Fig. 7 of the appendix.

To cope with the larger problems, we revise the condition for judging the payoff fluctuation of a classifier. The condition given in section IV is that $cl_i.p_{\max} - cl_i.p_{\min} < \theta_p$ and $cl_i.exp > \theta_t$, and if this condition is satisfied, XCSAT judges that the payoffs of the classifier does not fluctuate. Since the difference between $cl_i.p_{\max}$ and $cl_i.p_{\min}$ depends on the size of a problem and the payoff of a classifier decreases according to the discount factor γ every period of time, by using the discount factor γ, we employ a modified condition $\gamma cl_i.p_{\max} < cl_i.p_{\min}$ instead of $cl_i.p_{\max} - cl_i.p_{\min} < \theta_p$. The remaining procedure for judging the payoff fluctuation is the same as before, and the value of γ is set at $\gamma = 0.9$ due to increase of the problem size. The system with the revised condition for judging the payoff fluctuation is denoted by XCSATγ. The number of trials is 20. The other experimental conditions and the parameters are the same as those in the computational experiment for the six woods environments given in section V-B.

In this computational experiment, the performances are evaluated by the average of 20 trials, and we define trials to be valid for measurement as follows: (i) the average steps from the start to the goal is not larger than 100; (ii) a trial, which consists of 8000 episodes, finishes in 5 hours or less.

No trial of XCSM and XCSMH satisfies the two conditions. In XCSAT and XCSATγ, 13 and 14 trials out of the 20 trials meet the conditions for **Lab1**, respectively, and 5 and 6 trials meet them for **LargeMaze**, respectively. That is, XCSM and XCSMH are no longer workable for larger scale maze problems such as **Lab1** and **LargeMaze**, while XCSAT or XCSATγ properly works in more than a half of the whole trials for **Lab1** and in a quarter of them for **LargeMaze**. From this result of the computational experiment together with the data shown in the previous subsections, it is understood that the proposed learning classifier systems with an internal action table, XCSAT and XCSATγ, demonstrate superior performance, compared to XCSM and XCSMH.

Since in **LargeMaze**, the numbers of all states and aliased states are large and there exist many possible routes from the starts to the goals, compared to **Lab1**, the number of valid trials of **LargeMaze** is smaller that of **Lab1**, and the steps taken by the agent from the starts to the goals in **LargeMaze** are larger than those of **Lab1**. The performances of XCSAT and XCSATγ are summarized in Table VIII in a way similar to Table VII. As seen in Table VIII, the data supports the superiority of the performance of XCSATγ compared to that of XSCAT, and then the modified condition for judging the payoff fluctuation to larger problems is shown to be effective.

VI. Conclusion

In this paper, we develop a learning classifier system with an internal action table (XCSAT) to deal with sequential decision problems in non-Markov environments. In XCSAT, controlling the internal register is separated from classifiers, and aliased states are perceived by detecting fluctuation of the payoffs received by classifiers. After recognizing the existence of aliased states, the environmental information and the corresponding update of the internal register are recorded in the internal action table as a rule for updating the internal register. XCSAT identifies the perceived aliased state by referring to the internal action table.

By performing computational experiments where 9 woods environments are used, we demonstrate the effectiveness of XCSAT. In particular, XCSAT works well for woods environments such that the number of states are about 20 and the fraction of aliased positions is about 30% as used in Lanzi [18] and Lanzi and Wilson [20].

The success probability of learning for the larger problems by the proposed classifier systems (XCSAT and XCSATγ) are not high, therefore further improvement of the classifier system should be a future work.

References

[1] A. Burisov and A. Vasilyev, "Learning classifier systems in autonomous agent control task," Proceedings of the 5th International Conference on Application on Fuzzy Systems and Soft Computing, 36–42, 2002.

[2] L. Bull, J. Sha'Aban, A. Tomlinson, J. D. Addison and B. G. Heydecker, "Towards distributed adaptive control for road traffic junction signals using learning classifier systems," L. Bull (ed.), *Applications of Learning Classifier Systems*, Springer, New York, 279–299, 2004.

[3] M. V. Butz and S. W. Wilson, "An algorithm description of XCS," *Soft Computing*, 6, 144–153, 2002.

[4] Y. J. Cao, N. Ireson, L. Bull and R. Miles, "Design of a traffic junction controller using a classifier system and fuzzy logic," B. Reusch (ed.), *Computational Intelligence Theory and Applications*, Springer, New York, 342–353, 1999.

[5] D. Cliff and S. Ross, "Adding temporary memory to ZCS," *Adaptive Behavior*, 3, 101–150, 1994.

[6] M. Danek and R. E. Smith, "XCS applied to mapping FPGA architectures," *GECCO '02 Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers Inc., San Francisco, 912–919, 2002.

[7] K. De Jong, "Learning with genetic algorithms: an overview," *Machine Learning*, 3, 121–138, 1988.

[8] M. Dorigo and M. Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press/Bradford Books, Massachusetts, 1998.

[9] F. Doshi-Velez, J. Pineau and N. Roya, "Reinforcement learning with limited reinforcement: using Bayes risk for active learning in POMDPs," *Artificial Intelligence*, 187/188, 115–132, 2012.

TABLE VIII: Performance for larger problems (steps)

woods environment		XCSAT	XCSATγ	average of shortest steps
Lab1	valid trials	13/20	**14/20**	
number of all states: 79	mean	26.08	**20.99**	
number of aliased states: 35	best	15.59	**15.00**	14.50
	worst	61.22	**52.24**	
LargeMaze	valid trials	5/20	**6/20**	
number of all states: 148	mean	44.71	**37.13**	
number of aliased states: 100	best	31.37	**21.53**	14.00
	worst	55.41	**53.03**	

[10] A. Hamzeh, S. Hashemi, A. Sami and A. Rahmani, "A recursive classifier system for partially observable environments," *Fundamenta Informaticae*, 97, 15–40, 2009.

[11] J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Michigan, 1975. (reprinted by the MIT Press in 1992)

[12] J. H. Holland, "Adaptation," R. Rosen, F. M. Snell, (eds.), *Progress in Theoretical Biology*, 4, Academic Press, New York, 263–293, 1976.

[13] J. H. Holland, "Properties of the Bucket Brigade," J. J. Grefenstette (ed.), *Proceedings of the 1st International Conference on Genetic Algorithms*, L. Erlbaum Associates, New Jersey, 1–7, 1985.

[14] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, 313-329, 1978.

[15] J. H. Holmes and J. A. Sager, "Rule discovery in epidemiologic surveillance data using EpiXCS: an evolutionary computation approach," S. Miksch, J. Hunter and E. Keravnou (eds.), *Artificial Intelligence in Medicine*, Springer, Berlin, 444–452, 2005.

[16] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, 4, 237–285, 1996.

[17] P. L. Lanzi, "An analysis of the memory mechanism of XCSM," *Proceedings of the Third Genetic Programming Conference*, Morgan Kaufmann Publishers, San Fransisco, 643–651, 1998.

[18] P. L. Lanzi, "Adaptive agents with reinforcement learning and internal memory," J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (eds.), *From Animals to Animats 6, Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, MIT Press, Cambridge, 333–342, 2000.

[19] P. L. Lanzi, "Learning classifier systems from a reinforcement learning perspective," *Soft Computing*, 6, 162–170, 2002.

[20] P. L. Lanzi and S. W. Wilson, "Toward optimal classifier system performance in non-Markov environments," *Evolutionary Computation*, 8, 393–418, 2000.

[21] M. L. Littman, A. R. Cassandra and L. P. Kaelbling, "Learning policies for partially observable environments: scaling up," M. N. Huhns and M. P. Singh (eds.), *Readings in Agents*, Morgan Kaufmann Publishers, San Francisco, 495–503, 1998.

[22] J. Pineau, G. Gordon and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," mimeo, Carnegie Mellon University, 2003.

[23] R. J. Preen and L. Bull, "Discrete and fuzzy dynamical genetic programming in the XCSF learning classifier system," *Soft Computing* 18, 153–167, 2014.

[24] N. Roy, G. Gordon and S. Thrun, "Finding approximate POMDP solutions through belief compression," *Journal of Artificial Intelligence Research*, 23, 1–40, 2005.

[25] S. Saxon and A. Barry, "XCS and the Monk's problems," P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.), *Learning Classifier Systems: From Foundations to Applications*, Springer-Verlag, Berlin, 223–242, 2000.

[26] G. Shani, R. I. Brafman and S. E. Shimony, "Model-based online learning of POMDPs," J. Gama, R. Camacho, P. Brazdil, A. Jorge and L. Torgo (eds.), Machine Learning: ECML 2005, Springer-Verlag, Berlin, 353–364, 2005.

[27] W. Stolzmann, "Latent learning in Khepera robots with anticipatory classifier systems," A. Wu (ed.), *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program*, Morgan Kaufmann, San Francisco, 290–297, 1999.

[28] W. Stolzmann, "An introduction to anticipatory classifier systems," P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.), *Learning Classifier Systems: From Foundations to Applications*, Springer-Verlag, Berlin, 175–194, 2000.

[29] W. Stolzmann and M. V. Butz, "Latent learning and action planning in robots with anticipatory classifier systems," P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.), *Learning Classifier Systems: From Foundations to Applications*, Springer-Verlag, Berlin, 301–320, 2000.

[30] S. D. Whitehead and L. J. Lin, "Reinforcement learning of non-Markov decision processes," *Artificial Intelligence*, 73, 271–306, 1995.

[31] S. W. Wilson, "ZCS: a zeroth level classifier system," *Evolutionary Computation*, 2, 1–18, 1994.

[32] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, 3, 149–175, 1995.

[33] S. W. Wilson, "Mining oblique data with XCS," P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.), *Advances in Learning Classifier Systems*, Springer, Berlin, 158–176, 2001.

[34] S. W. Wilson, "Function approximation with a classifier system," *Proceedings of the genetic and evolutionary computation conference (GECCO '01)*, Morgan Kaufmann, San Francisco, 974–981, 2001.

[35] Z. Zang, D. Li, J. Wang and D. Xia, "Learning classifier system with average reward reinforcement learning," *Knowledge-Based Systems*, 40, 58–71, 2013.

APPENDIX

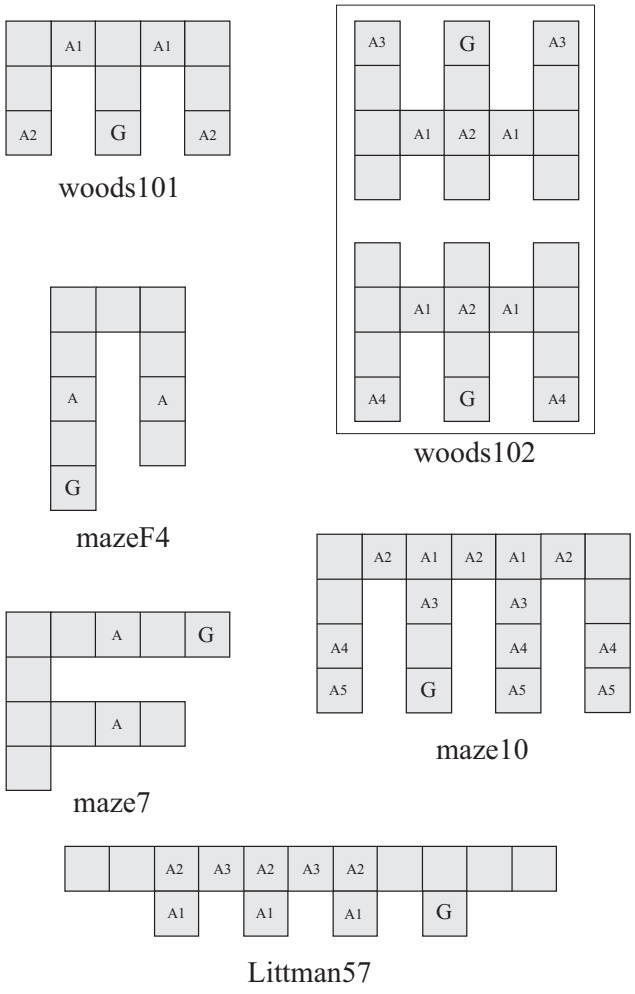

Fig. 6: Woods environments

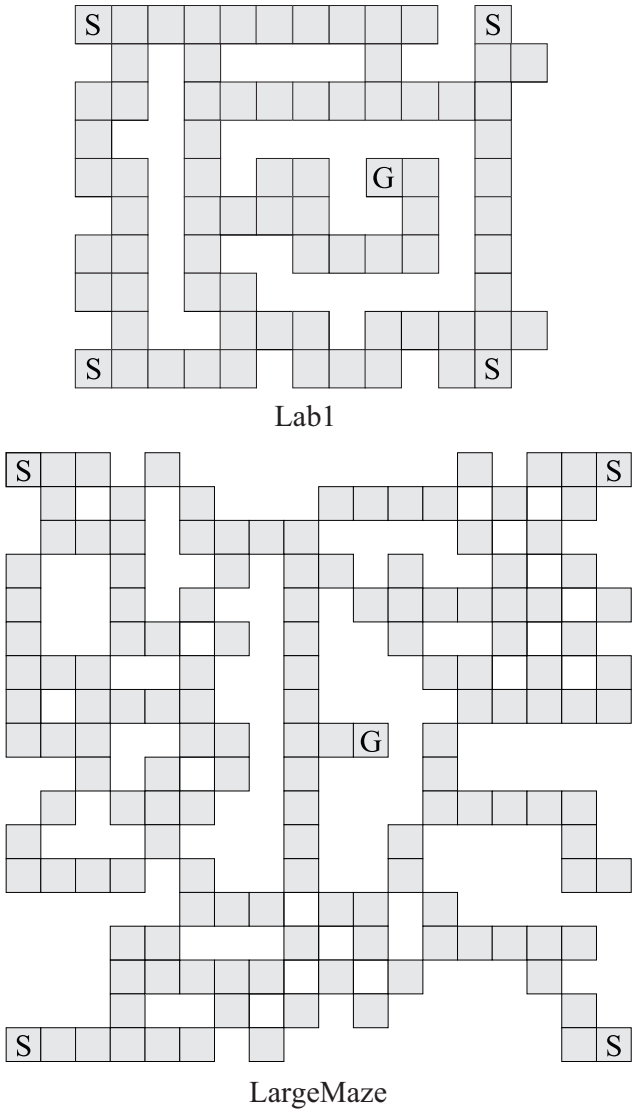

Fig. 7: Large woods environments