Using Learning Classifier Systems to Learn Stochastic Decision Policies

Gang Chen, *Member, IEEE,* Colin I J Douch, and Mengjie Zhang, *Senior Member, IEEE*

***Abstract*—To solve reinforcement learning problems, many learning classifier systems are designed to learn state-action value functions through a compact set of maximally general and accurate rules. Most of these systems focus primarily on learning deterministic policies by using a greedy action selection strategy. However, in practice, it may be more flexible and desirable to learn stochastic policies, which can be considered as direct extensions of their deterministic counterparts. In this paper, we aim to achieve this goal by extending each rule with a new policy parameter. Meanwhile, a new method for adaptive learning of stochastic action selection strategies based on a policy gradient framework has also been introduced. Using this method, we have developed two new learning systems, one based on a regular gradient learning technology and the other based on a new natural gradient learning method. Both learning systems have been evaluated on three different types of reinforcement learning problems. The promising performance of the two systems clearly shows that learning classifier systems provide a suitable platform for efficient and reliable learning of stochastic policies.**

***Index Terms*—Learning systems, Stochastic systems, Gradient methods**

I. Introduction

Learning classifier systems (LCSs) are evolutionary machine learning technologies originally introduced by John Holland [22], [23]. They are designed to learn decision-making policies jointly defined by a set of classifiers. Each classifier typically assumes the form of a "*condition-action-payoff*" rule. LCSs can be largely classified into two major types, namely Michigan LCSs and Pittsburgh LCSs [28]. In this paper, we will study accuracy-based Michigan LCSs, in particular the XCS classifier system [50].

Recently, the successful application of LCSs on many real-world problems has triggered increasing research interests [3], [5], [39], [42], [48]. In particular, LCSs have been frequently applied to tackle reinforcement learning problems with prominent success [45]. For example, XCS and relevant LCSs have been extensively explored to solve various robotic control problems with proven effectiveness [2], [43], [44]. Meanwhile, other miscellaneous applications such as chemical reaction control [7], traffic control [15], etc. have also clearly demonstrated the usefulness of LCSs as effective machine learning tools.

Gang Chen, Colin Douch, and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand (e-mail: aaron.chen@ecs.vuw.ac.nz; douchcoli@myvuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

To the best of our knowledge, most of the XCS-based systems attempted to solve a reinforcement learning problem by learning a *state-action value function*. Such a value function is jointly approximated by multiple classifiers in an LCS and is exploited to foretell the expected long-term payoff of performing any action in every possible state. Very often, the payoff is defined as the current reward plus the discounted future reward to be received by a learning agent [18], [47]. Driven by this value-function based approach, by further utilizing a proper *action selection strategy*, the agent is expected to solve a learning problem and hereby achieve the maximum possible payoff.

To meet this goal, many existing research works utilized a greedy action selection strategy. It defines a *deterministic policy* that maps every state to the best action to be performed in that state in order to maximize the state-action value function. In fully observable and deterministic environments, there exists at least one optimal deterministic policy that is obtainable by using the greedy strategy. Nevertheless, when the environment is stochastic and partially observable, no deterministic policy may enable a learning agent to achieve its maximum payoff [41]. In comparison with a fully observable environment, a partially observable environment is marked by the absence of the *Markov property*, i.e. the observation from the current problem state cannot completely determine the probability of reaching any new states upon performing an action. Due to this observational limitation, a learning agent may be trapped within a local loop or local optima whenever a deterministic policy is adopted, therefore failing to achieve its learning objective [41].

Instead of learning deterministic policies, an agent that learns *stochastic policies* may easily escape from a local loop and hence potentially present a good solution to the above problem [19]. This idea can be realized by using, for example, a *probabilistic action selection strategy* where the chance of performing any action in a state is made proportional to its expected payoff (i.e. according to the state-action value function). Unfortunately, as will be illustrated in Section IV, this strategy may fail to produce any optimal policy. The ϵ-*greedy* action selection strategy is another approach for constructing stochastic policies. It is very often used during learning where an action is either selected uniformly at random or the best action identified so far according to the learned state-action value function will be performed. However, the ultimate goal is still to learn deterministic policies. The ϵ-*greedy* method itself may not produce any optimal policy.

To achieve a high level of flexibility in learning, we aim at developing new mechanisms for direct learning of stochastic

action selection strategies in this paper. In XCS, action selection depends heavily on the value function. Without using the learned value function, e.g. by removing the prediction parameter from each classifier, exploitive action selection (or similar action selection strategies) during the testing phase will not work properly. In this paper, each classifier in an LCS is expanded to include a new *policy parameter* that directly controls the probability of performing each action. While testing the performance of the learned classifiers, we can safely remove their prediction parameters without causing any performance degradation. As discussed in [46], this separation has an immediate benefit that arbitrarily small updates to the value function will not abruptly change an agent's preference of taking any action.

Using policy parameters enables us to learn arbitrary stochastic policies, including any deterministic policy as a special instance. A policy gradient framework has also been utilized in this paper to guide the learning of policy parameters. This is achieved with the development of a new *policy parameter learning component*. Two alternative learning rules have been created for this component, one based on a regular gradient learning technology and the other based on a new natural gradient learning method.

Powered by the two learning rules, two new LCSs have been developed based on XCS in this paper, known respectively as Regular gradient XCS (i.e. RXCS) and Natural XCS (i.e. NXCS). In order to evaluate their learning performance, experiments have been conducted on three different types of reinforcement learning problems. Specifically, we found that both NXCS and RXCS perform very well in partially observable environments. The effectiveness of the two learning systems is also clearly evident in environments where the outcome of performing any action is subject to high level of randomness (see Subsection VII-B). Meanwhile, NXCS and RXCS can effectively solve traditional benchmark maze problems where the optimal policies are deterministic. The promising results clearly show that LCSs provide a suitable platform for efficient and reliable learning of stochastic policies.

The remainder of this paper is organized as follows. Section II reviews related works. A short introduction to reinforcement learning and the XCS classifier system can be found in Section III. Based on XCS, NXCS and RXCS will be further developed in Section IV, Section V, and Section VI. The performance of the two new learning systems will be experimentally studied in Section VII. Section VIII concludes this paper and points out possible future research directions.

II. Related work

A. LCSs for reinforcement learning

The introduction of XCS by Wilson in 1995 marked a milestone in learning classifier system research [50]. According to [28], XCS is the first classifier system that is both general enough for a wide range of applications and simple enough for in-depth analysis. Years of research also revealed some limitations of XCS in solving difficult reinforcement learning problems. For example, XCS failed to solve some problems studied in [10] where a large number of actions have to be performed until reinforcement is encountered. XCS was also shown to be ineffective when the learning environment is stochastic or partially observable [27], [29], [32].

As the understanding of XCS and other LCSs deepened in the last decade, substantial improvements have also been made to build new learning systems with enhanced capability and performance [2], [10], [27], [28], [33], [40], [47]. For example, researchers have put in huge efforts to develop flexible representations of classifiers, including their conditions, actions, and prediction functions, in order to enhance the applicability of LCSs in many real-life learning tasks [9], [12], [13], [30], [31]. Similar efforts have also led to the wide exploitation of many artificial intelligence (AI) technologies, such as fuzzy logic, neural network, and genetic programming [8], [16], [38], [40].

In particular, technologies for learning fuzzy classifiers in XCS-based systems have received considerable research attention [16], [17], [21]. In a Fuzzy-XCS system [16], an action can be directly obtained (in a deterministic manner) from a selected *action group* of fuzzy classifiers that match any environmental input. The action group with the highest mean prediction will be further used to generate the output action [16], [17]. For this purpose, multiple groups of consistent and non-redundant fuzzy classifiers will be formed in the first place. It is interesting to note that, in order to avoid the ordering bias among all fuzzy classifiers in the learning system, some random mechanisms have been adopted in the group formation process. Because of that, the output action is stochastic in nature. However, generating random actions was not the main focus and was not explicitly controlled by any learning technique in [16], [17]. Another major difference from this paper lies in the fact that the fuzzy systems in [16], [17] were designed to address single-step problems. On the other hand, our target is multiple-step reinforcement learning problems.

Motivated by the understanding that the classifiers and their parameters in XCS jointly approximate the state-action value function that solves a reinforcement learning problem, Butz *et al.* proposed a new gradient descent approach for learning these parameters [10]. As a result, a new learning system called *XCS with Gradient Descent* was created with demonstrated effectiveness. Meanwhile, to improve the effectiveness of XCS when a learning environment is stochastic or partially observable, XCS-based systems, such as XCS_μ [29], have also been proposed in the literature. As explained by Lanzi and Colombetti, "the minimum prediction error experienced in each environmental niche can be used to evaluate the inaccuracy introduced by the environment" [29]. Driven by this understanding, the updating of the prediction error in XCS_μ is revised by taking into account the inaccuracy due to the uncertainty on agents' actons.

Similar to the research works mentioned above, this paper also features gradient descent methods for learning. Nevertheless, the goal of XCS with Gradient Descent is to learn a good approximation of the state-action value function. On the other hand, this paper focuses primarily on learning suitable stochastic action selection strategies. Meanwhile, in this paper, we have no interest in addressing the potential erroneous

update of prediction error. Instead, we believe a policy gradient method, specifically a natural gradient learning technique, can be more reliable at handling environmental randomness (see Section VII for experimental comparison between XCS$_\mu$ and our learning systems).

B. Policy gradient learning methods

In the literature, both the policy gradient methods and the natural gradient learning techniques have been popularly researched for reinforcement learning [6], [25], [36], [46]. Different from value function based learning algorithms, policy gradient methods explicitly represent a decision-making policy through a parametric model. Any change to the parameters in the model by a learning agent may lead to behavioral alterations. The agent can hence produce many different solutions to a learning problem.

Depending on the nature of the model, the policies to be learned by the agent can be either deterministic or stochastic [20]. Researchers have explored many different model representations, ranging from simple linear representations to more sophisticated time-dependent representations [24]. This paper will demonstrate the use of a different representation technique where the policy is modelled through a set of dynamically evolving classifiers.

After determining the models to be used for representing a policy, the next step is to develop useful search techniques in the parametric policy space. For this purpose, many different policy gradient search methods have been proposed [25], [35], [46], [49]. In practice, the policy gradient is used to quantify the influence of each policy parameter on the overall learning performance. It essentially determines the scale and direction of each learning step. With the help of the policy gradient, effective search in the policy space can be performed towards identifying the optimal policies in the learning problem.

In the literature, the finite difference policy gradient is widely considered as a simple way to obtain the policy gradient [26], [35]. Besides that, likelihood-ratio methods have also been developed to determine policy gradients and guide the search in the policy space [49]. Because a sampling process is often used to determine a local estimation of the policy gradient, the next gradient estimation may change dramatically. As a result, the learning effectiveness will suffer. To address this issue, it is desirable to enforce reasonably small change in each learning step. The natural gradient concept has been introduced to achieve this goal [1], [20].

Since 2000, this natural gradient learning technique has attracted increasing research interests and several natural reinforcement learning algorithms have been successfully developed to facilitate the learning of stochastic policies [6], [25], [36]. In addition to promoting learning stability, these algorithms were clearly shown to be reliable tools for many robot reinforcement learning tasks [34], [37].

To summarise, we found that policy gradient methods have been extensively studied within the context of traditional reinforcement learning algorithms. However, according to our knowledge, few research works on evolutionary reinforcement learning algorithms, such as the LCSs, have utilized policy gradient techniques. To fill this gap, in this paper, we will embed policy gradient mechanisms into an evolutionary system for reinforcement learning and study their effectiveness.

III. Reinforcement Learning and the XCS Classifier System

A reinforcement learning problem is commonly defined as a problem where an agent learns to perform a task through *trial-and-error interaction* with an unknown environment [45]. At any time t, the agent is in a state s_t that belongs to the set S of all possible states of the environment. It senses its environment and selects an action a_t among the set A of possible actions. The action is then performed in the environment. Subsequently, the agent receives a *scalar reward* r_{t+1} as the feedback from the environment. It also changes to a new state s_{t+1}. Starting from an arbitrary state of the environment, the agent's goal is to *maximize* the *discounted expected payoff* it receives in the long run, which at time t is defined as below:

$$E\left[\sum_{k=0}^{\infty}\gamma^k \cdot r_{t+1+k}\right] \tag{1}$$

where γ ($0 \leq \gamma \leq 1$) is the *discount factor*. XCS can be utilized to solve this learning problem. In particular, by learning the state-action value function $Q(s,a)$ through a *population* $[P]$ of classifiers, the greedy action selection strategy will be further exploited to identify a solution to the problem in the form a deterministic policy. The readers are referred to [14] for a detailed algorithmic description of XCS. A brief introduction to XCS is presented below by reviewing its four key components. The use of the four components for learning is also illustrated later in Fig. 2.

A. Classifiers

In XCS, each classifier cl includes a condition c_{cl} and an action a_{cl}. It also has seven parameters, including particularly the prediction p_{cl} that estimates the average payoff to be expected whenever the classifier is used; the fitness F_{cl} that estimates the average relative accuracy of classifier cl; the experience exp_{cl} that counts the number of times that classifier cl has participated in an *action set*; and the numeriosity num_{cl} that indicates how many classifiers represented by classifier cl are present in the population.

B. Performance component

At any time t, XCS creates a *match set* $[M]_t$ containing all classifiers in the population that match the current sensory input from state s_t. For every action $a \in A$, the *system prediction* $P_t(a)$, which is a local approximation of $Q(s_t,a)$, will be determined according to (2):

$$Q(s_t,a) \approx P_t(a) = \frac{\sum_{cl\in[M]_t^a} p_{cl} \times F_{cl}}{\sum_{cl\in[M]_t^a} F_{cl}} \tag{2}$$

where $[M]_t^a$ refers to a collection of those classifiers in $[M]_t$ that recommend action a. During learning, the ϵ-*greedy*

action selection method will be employed. Specifically, for all experiments reported in Section VII, there is 50% chance of selecting an action a_t according to (3) [14]. On the other hand, during testing, the greedy action selection strategy described also by (3) will be adopted. Once an action a_t is selected and performed, the *action set* $[A]_t$ will be further constructed for the subsequent reinforcement component.

$$a_t = \underset{a}{\text{argmax}} P_t(a) \tag{3}$$

C. Reinforcement component

Upon reaching a new state s_{t+1} at time $t+1$, the agent will receive a scalar reward r_{t+1} from the environment. Accordingly, the parameters of those classifiers in $[A]_t$ will be updated based on the updating rules in [14]. In particular, with a fixed learning rate β ($0 \leq \beta \leq 1$), the rule for updating the prediction p_{cl} of a classifier $cl \in [A]_t$ is given below:

$$p_{cl}(t+1) \leftarrow p_{cl}(t) + \beta \left(r_{t+1} + \gamma \max_{a \in A} P_{t+1}(a) - p_{cl}(t) \right) \tag{4}$$

D. GA component

On a regular basis, a genetic algorithm will be applied to those classifiers in $[A]_t$. In particular, two classifiers from $[A]_t$ will be selected for reproduction with probability proportional to their fitness. The chosen classifiers are then copied to produce offspring classifiers. Before joining the population $[P]$, the offspring are first combined through a crossover operation with probability χ and further modified through a mutation operation with probability μ.

IV. Building a stochastic policy

As shown in (2), classifiers in XCS are designed to approximate the state-action value function $Q(s,a)$. The greedy action selection strategy defined in (3) subsequently produces a deterministic policy that solves a reinforcement learning problem. However, as proven by Singh *et al.* in [41], in stochastic and partially-observable environments, the best stochastic policy can be *arbitrarily better than* the best deterministic policy. This means that the expected payoff obtainable from learning a stochastic policy can be much higher than the payoff achievable by using any deterministic policy (at least on carefully engineered problems).

In practice, by following the *probabilistic action selection strategy*, a stochastic policy can be directly derived from function $Q(s,a)$. As a simple example, when the agent is in state s_t at time t, the probability of performing any action a can be calculated from

$$\pi_t(s_t, a) = \frac{Q(s_t, a)}{\sum_{a' \in A} Q(s_t, a')} \tag{5}$$

XCS that uses (5) for action selection will be called *XCS with Roulette Wheel Action Selection* (XCSrwas) in the sequel. Unfortunately, XCSrwas may not be able to learn optimal policies, even for single-step problems. For instance, Fig. 1 shows a simple reinforcement learning problem where the goal for an agent is to reach state F from state S by performing either action a_1 or a_2. Obviously, the best strategy for the agent is to always perform action a_1 in state S. The expected reward in this case is 1000. However, when (5) is employed for action selection, the probability for the agent to perform a_1 is only 2/3. The expected reward is also reduced to 833.3. It is further confirmed by the experiment results reported in Section VII that XCSrwas is ineffective on all benchmark problems studied in this paper.

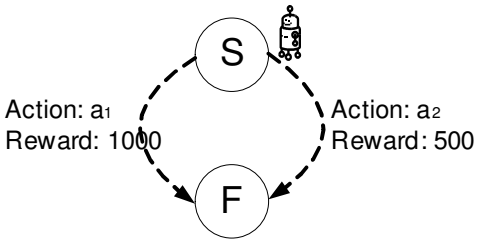

Fig. 1. A simple single-step learning problem with two states S and F and two actions a_1 and a_2. The learning agent is in state S.

Obviously, (5) can be useful in some situations. For example, when the reward of performing action a_2 in Fig. 1 is increased to 1000, then (5) will actually produce an optimal stochastic policy. Our simple example clearly shows that, by using the state-action value function alone, it is difficult for an agent to decide when to use a stochastic policy or a deterministic one. To achieve high level of learning flexibility, the use of an action selection strategy should not depend heavily on the state-action value function. For this purpose, in this paper, each classifier is extended with an additional *policy parameter*, denoted as ω_{cl}. At any time t, using all classifiers that belong to the match set $[M]_t$, the probability of choosing any action $a \in A$ is decided according to (6) below.

$$\pi_t(s_t, a) = \frac{\prod_{cl \in [M]_t^a} e^{\omega_{cl}}}{\sum_{b \in A} \left(\prod_{cl \in [M]_t^b} e^{\omega_{cl}} \right)} \tag{6}$$

As evident in (6), stochastic action selection in this paper is performed independent of the value function Q. A similar formula like (6) is also utilized by the softmax action selection method [45], which however relies on Q instead of the policy parameter ω_{cl} to determine the probability of performing any action.

Continuing with the example in Fig. 1, assume that the agent is in state S at time t. Also assume that each of the two alternative actions, i.e. a_1 and a_2, is associated with exactly one classifier in $[M]_t$, represented respectively as cl_1 and cl_2. Then, the probability of performing a_1 and a_2 are

$$\pi(S, a_1) = \frac{e^{\omega_{cl_1}}}{e^{\omega_{cl_1}} + e^{\omega_{cl_2}}} \text{ and } \pi(S, a_2) = \frac{e^{\omega_{cl_2}}}{e^{\omega_{cl_1}} + e^{\omega_{cl_2}}}$$

respectively. Using exponential functions in the above equations makes it easy for us to closely approximate the optimal

deterministic policy of selecting action a_1. As a matter of fact, when $\omega_{cl_1} = 10.0$ and $\omega_{cl_2} = -10.0$, numerical calculation of $\pi(S, a_1)$ will produce a probability of 0.999999998. Because of this, in Section VII, the value range for all policy parameters is set to $(-10, +10)$. Clearly, by adjusting ω_{cl_1} and ω_{cl_2}, arbitrary probability of performing either a_1 or a_2 is possible. Hence, in theory any stochastic policy (including deterministic policies) can be obtained by learning the policy parameters.

For this reason, a policy parameter vector $\vec{\omega}_t$ is further constructed to include ω_{cl} of every classifier cl in the population $[P]_t$. Obviously, (6) is not the only way of defining a stochastic policy (or stochastic action selection strategy). However, it is frequently used for building statistical models [6], [46]. In [6], [46], $\vec{\omega}_t$ corresponds to a fixed collection of state-action features which will remain unchanged during learning. On the contrary, in this paper, every ω_{cl} is provided by a classifier cl. As classifiers are evolving during learning, $\vec{\omega}_t$ is also changing dynamically over time.

One benefit of using (6) is that $\nabla_{\vec{\omega}_t} \pi_t(s_t, a_t)$, which will be used later for learning policy parameters, can be easily calculated from (7) below.

$$\nabla_{\vec{\omega}_t} \pi_t(s_t, a_t) = \left(\pi_t(s_t, a_t) - \pi_t(s_t, a_t)^2\right) \vec{\phi}_{a_t,t} \tag{7}$$

Meanwhile, $\nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t)$, which will also be utilized for learning policy parameters, can be obtained directly from

$$\nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t) = \vec{\phi}_{a_t,t} - \sum_{b \in A} \pi_t(s_t, b) \cdot \vec{\phi}_{b,t} \tag{8}$$

where, for any action $a \in A$, $\vec{\phi}_{a,t}$ is defined as below

$$\vec{\phi}_{a,t}(k) = \begin{cases} 1, & cl_k \in [M]_t^a \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$\vec{\phi}_{a,t}(k)$ in (9) stands for the k-th element of $\vec{\phi}_{a,t}$, $1 \leq k \leq N$. N is the dimension of $\vec{\omega}_t$. Using the example in Fig. 1 again, suppose that there are only two classifiers cl_1 and cl_2 (as we introduced earlier) in the population $[P]$ when the agent is in state S. Then $\vec{\phi}_{a_1} = \{1, 0\}$, indicating the fact that only classifier cl_1 in $[P]$ matches state S and recommends action a_1. Similarly, we can also see that $\vec{\phi}_{a_2} = \{0, 1\}$.

V. An algorithmic extension of XCS for learning stochastic policies

XCS presents an effective approach for learning a complete, accurate, and maximally general set of classifiers [11]. The same learning procedure will also be utilized to learn stochastic policies in this paper. However, some modifications and extensions, as we explained below, are necessary.

- Action selection during learning no longer needs to follow the ϵ-*greedy* method. Instead, since exploration is directly supported by the stochastic policy being learned, in every state encountered, an action will be chosen with the probability given in (6).
- At any time t, the prediction of each classifier $cl \in [A]_t$, i.e. p_{cl}, will be updated according to a new updating rule presented in (10) below. In comparison with the updating rule of XCS, as described in (4), change is introduced in (10) because prediction cannot be updated by assuming that the action chosen by (3) will be performed at time $t + 1$. Instead, any action a may be performed with a probability of $\pi_t(s_{t+1}, a)$.
- To adjust the probability of performing any action in response to the rewards received from the environment, the policy parameters need to be constantly updated. For this purpose, a separate policy parameter learning component is introduced into the learning process. After updating ordinary parameters of relevant classifiers, their policy parameters will also be updated based on either a regular gradient learning method or a new natural gradient learning method.

$$p_{cl}(t+1) \leftarrow p_{cl}(t) + \beta \cdot \left(r_{t+1} + \gamma \sum_{a \in A} \pi_t(s_{t+1}, a) \cdot P_{t+1}(a) - p_{cl}(t)\right) \tag{10}$$

```
While (termination criteria not met) {
  Update current time t
  Obtain sensory input from current state s_t
  Generate match set [M]_t
  Calculate the probability of taking any action based on (6)
  Select an action according to the calculated probability
  Perform selected action
  Observe reward r_{t+1} and state s_{t+1}
  For every classifier cl that belongs to [A]_t {
    Update prediction p_cl according to (10)
    Update other parameters of cl by using the same updating rules
    in XCS
  }
  Execute the policy parameter learning component
  Run GA component on [A]_t
  Delete classifiers when the population is full
}
```

Fig. 2. An algorithmic extension of XCS for learning stochastic policies.

As we described in Fig. 2, the iterative learning process in XCS is directly extended to learn stochastic policies. In fact, except for the prediction parameter, other parameters of every classifier $cl \in [A]_t$ will be updated by following the same updating rules employed in XCS. After that, the policy parameter learning component will be activated to update the policy parameters of relevant classifiers. Details of this component will be presented in Section VI. The ordinary GA component of XCS will be applied afterwards and learning will then proceed to the next iteration.

Since the learning procedure shown in Fig. 2 is almost identical to that of XCS, we are expected to maintain many of the strengths of XCS while learning stochastic policies. For example, because the evolution of classifiers is driven by the same accuracy-based fitness updating rule and the same GA component as in XCS, our extended learning system will not reduce XCS's capability of learning accurate and general classifiers. This observation is verified experimentally in Section VII.

VI. Developing a policy parameter learning component

In this section, we aim at developing a new policy parameter learning component. For this purpose, a mathematical connection between learning performance and the policy parameters should be established in the first place. In line with (1), the performance of using any policy $\pi(s,a)$ by a learning agent can be measured in the form of the discounted cumulative reward in the long run [46], as shown below.

$$J(\pi) = \sum_{s\in S} d^{\pi}(s) \sum_{a\in A} \pi(s,a)\cdot R(s,a) \tag{11}$$

$R(s,a)$ above stands for the expected reward to be received from the environment when action a is performed in state s. $d^{\pi}(s)$ is commonly known as the *stationary probability* [46] and is defined below.

$$d^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t Pr^{\pi}(s_t = s)$$

where $Pr^{\pi}(s_t = s)$ represents the probability for the agent to be in state s at any time t upon using policy π. Because $\pi(s,a)$ is a function of $\vec{\omega}$, it is straightforward to see that the performance J is also a function of $\vec{\omega}$. Consequently, the agent's learning objective is to find the best $\vec{\omega}^*$ as defined below

$$\vec{\omega}^* = \underset{\vec{\omega}}{\operatorname{argmax}} J(\vec{\omega}) \tag{12}$$

A. A policy gradient framework

In order to develop a learning component for policy parameters, a policy gradient framework is adopted in this paper. In comparison with the more well-established value-function based approaches for reinforcement learning, existing research works showed that policy gradient methods are highly competitive when an agent can only learn approximated solutions [4], [6], [46]. Given the learning objective in (12), a straightforward approach is to learn $\vec{\omega}$ based on

$$\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \lambda_t \cdot \nabla_{\vec{\omega}_t} J \tag{13}$$

where λ_t is the *learning rate*. (13) clearly shows the direction of learning which is determined completely by $\nabla_{\vec{\omega}_t} J$. Using unbiased estimation of $\nabla_{\vec{\omega}_t} J$ and under suitable assumptions of λ_t, the policy parameter learning process, as described in (13), is stable and will eventually converge to $\vec{\omega}^*$ [46]. This convergence result can potentially be generalized to include the case when the *natural gradient* is exploited to update $\vec{\omega}$ [6]. Learning through natural gradient will be presented later in Subsection VI-C.

B. Learning policy parameters based on a regular gradient method

Following (13), for the purpose of learning $\vec{\omega}$, we need to find a way of approximating $\nabla_{\vec{\omega}_t} J$. As explained in [46], this requires us to use the *advantage function* defined below

$$A(s,a) = Q(s,a) - V(s) \tag{14}$$

for any possible state s and possible action a. The value function $V(s)$ in (14) is further defined as

$$V(s) = \sum_{a\in A} \pi(s,a)\cdot Q(s,a) \tag{15}$$

Given the fact that unbiased approximation of the state-action value function $Q(s,a)$ is learned by XCS and can be computed directly from (2), at any time t during learning, unbiased approximation of $V(s_t)$ and $A(s_t,a_t)$ can also be computed straightforwardly. In particular, $A(s_t,a_t)$ can be approximated by δ_t given in (16).

$$\delta_t = r_{t+1} + \gamma \cdot \sum_{a\in A} \pi_t(s_{t+1},a)\cdot P_{t+1}(a) - \sum_{a\in A} \pi_t(s_t,a)\cdot P_t(a) \tag{16}$$

According to [46], unbiased approximation of $\nabla_{\vec{\omega}_t} J$ can be further obtained by using (17) below.

$$\nabla_{\vec{\omega}_t} J = \sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \nabla_{\vec{\omega}_t} \pi_t(s,a)\cdot A(s,a) \tag{17}$$

Based on (17), to update $\vec{\omega}$, we may have to go through all possible states and actions. Fortunately, at any time t, only those classifiers belonging to the match set $[M]_t$ will be utilized for action selection. Moreover, after performing the selected action a_t, only the prediction of those classifiers in the action set $[A]_t$ will be updated according to (10). Therefore, it is unnecessary for us to update the policy parameter of every classifier in the population but only those in $[A]_t$. From (17), for any classifier $cl \in [A]_t$, the derivative of J with respect to ω_{cl} can be approximated easily as

$$\frac{\partial J}{\partial \omega_{cl}} \approx d^{\pi_t}(s_t)\cdot \delta_t \cdot \frac{\partial \pi(s_t,a_t)}{\partial \omega_{cl}} \tag{18}$$

Assuming that the stationary probability $d^{\pi_t}(s_t)$ is a constant, an updating rule for learning policy parameters is subsequently constructed as below

$$\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \lambda \cdot \delta_t \cdot \nabla_{\vec{\omega}} \pi(s_t,a_t) \tag{19}$$

where $\vec{\omega}$ only contains the policy parameters of those classifiers in $[A]_t$. $\nabla_{\vec{\omega}} \pi(s_t,a_t)$ is given in (7). Based on (19), a new LCS known as the Regular gradient XCS (i.e. RXCS) is created. RXCS follows the learning process described in Fig. 2. Fig. 3 further explains how the policy parameters are updated in RXCS.

C. Learning policy parameters based on a natural gradient method

Practical application often shows that learning through the regular gradient, i.e. (13), can be slow and unstable [1], [36]. Instead of using $\nabla_{\vec{\omega}_t} J$, a *natural gradient* concept proposed by Amari can be very helpful [1]. Theoretically, stochastic policies learned through a classifier system are equivalent to a family of statistical models situated in a parameter

```
LearnPolicyParameters() {
    Update current time t.
    For every possible action a ∈ A {
        Calculate P_t(a) and P_{t+1}(a) according to (2).
    }
    Calculate δ according to (16).
    For every classifier cl ∈ [A]_t {
        Calculate ∇_{ω⃗_cl} π(s_t, a_t) according to (7).
        Update ω_cl according to (19).
    }
}
```

Fig. 3. The policy parameter learning component in RXCS.

vector space of $\vec{\omega}$. Each point in the space corresponds to a specific stochastic policy. In a *Riemannian space*, which is an extension of the *Euclidean space*, the distance between any two neighboring points, namely $\vec{\omega}$ and $\vec{\omega} + \vec{d}$, is defined as

$$\|\vec{d}\| = \sqrt{\vec{d}^T \cdot G(\vec{\omega}) \cdot \vec{d}} \tag{20}$$

where $G(\vec{\omega})$ is a $N \times N$ positive-definite matrix known as the *Riemannian metric tensor*. If $G(\vec{\omega})$ is always an *identity matrix*, then the Riemannian space reduces to an Euclidean space. In general, however, G is a function of $\vec{\omega}$. In the Euclidean space, learning of $\vec{\omega}$ is carried out through the updating rule in (19). In the Riemannian space, on the other hand, according to [1], learning should be performed based on the *natural gradient* of J, i.e. $\tilde{\nabla}_{\vec{\omega}_t} J$. Particularly, we have

$$\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \lambda \cdot \tilde{\nabla}_{\vec{\omega}_t} J \tag{21}$$

where the natural gradient $\tilde{\nabla}_{\vec{\omega}_t} J$ gives the steepest ascent direction of J in the Riemannian space of $\vec{\omega}$ and is defined as

$$\tilde{\nabla}_{\vec{\omega}_t} J = G(\vec{\omega}_t)^{-1} \cdot \nabla_{\vec{\omega}_t} J \tag{22}$$

The choice of $G(\vec{\omega}_t)$ seems to be arbitrary. In practice, the *Fisher information matrix* presented below is often used as the Riemannian metric tensor.

$$G(\vec{\omega}) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi(s,a) \nabla_{\vec{\omega}} \log \pi(s,a) \cdot \nabla_{\vec{\omega}} \log \pi(s,a)^T \tag{23}$$

According to [1], for general statistical learning tasks, using the Fisher information matrix can achieve the optimal learning efficiency. For this reason, we will use (23) to define natural gradient in this paper. Another important benefit of using (23) is that $\tilde{\nabla}_{\vec{\omega}_t} J$ can be approximated efficiently by the policy parameter learning component, even without the need of calculating $G(\vec{\omega})$ or $G(\vec{\omega})^{-1}$. Specifically, similar with the deductions presented in [6], [25], it is shown in Appendix A that, for $\vec{\omega}_t$ that includes only the policy parameters of those classifiers in $[M]_t$, $\tilde{\nabla}_{\vec{\omega}_t} J$ is approximately proportional to

$$\tilde{\nabla}_{\vec{\omega}_t} J \quad \tilde{\propto} \quad \vec{w} + \lambda \cdot \delta_t \cdot \nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t) - \lambda \cdot \nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t) \cdot \nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t)^T \cdot \vec{w}$$

where $\nabla_{\vec{\omega}} \log \pi_t(s_t, a_t)$ is given in (8). $\vec{w}$ refers to an initial approximation of $\tilde{\nabla}_{\vec{\omega}_t} J$. By simply setting $\vec{w}$ to $\vec{0}$, the updating rule for learning policy parameters is immediately obtained as

$$\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \lambda \cdot \delta_t \cdot \nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t) \tag{24}$$

```
LearnPolicyParameters() {
    Update current time t.
    For every possible action a ∈ A {
        Calculate P_t(a) and P_{t+1}(a) according to (2).
    }
    Calculate δ according to (16).
    For every classifier cl ∈ [M]_t {
        Calculate ∇_{ω⃗_cl} log π(s_t, a_t) according to (8).
        Update ω_cl according to (24).
    }
}
```

Fig. 4. The policy parameter learning component in NXCS.

The learning system that uses (24) is called the Natural XCS (i.e. NXCS) classifier system in this paper. Fig. 4 describes how policy parameters are learned in NXCS. According to (24), the amount of update applied to $\vec{\omega}_t$ is partially determined by δ_t, which is closely related to the reward r_{t+1}. To ensure that parameter update will not be strongly influenced by the level of reward that the environment can generate, a direct method is to set λ to the reciprocal of the maximum absolute value of the single-step reward. This approach is adopted by both NXCS and RXCS.

D. Understanding the learning rules

In (19) and (24), two learning rules have been successfully developed for updating the policy parameters at any time t. The practical meaning of these two rules can be easily interpreted. Specifically, according to (19), whenever $\delta_t > 0$, $\vec{\omega}_t$ will be updated in the direction of increasing $\pi(s_t, a_t)$, which is the probability of performing action a_t in state s_t. By further checking the definition of δ_t in (16), it can be immediately seen that $\delta_t > 0$ as long as the payoff of performing action a_t, which is

$$r_{t+1} + \gamma \cdot \sum_{a \in A} \pi_t(s_{t+1}, a) \cdot P_{t+1}(a),$$

is greater than the expected payoff obtainable from following policy π_t in state s, that is

$$\sum_{a \in A} \pi_t(s_t, a) \cdot P_t(a).$$

In other words, we should increase the chance of performing action a_t if it can produce better-than-average payoff. On the other hand, a learning agent will become less likely to select action a_t if it gives rise to worse-than-average payoff (i.e. $\delta_t < 0$). This is exactly what the agent is supposed to do, i.e. maximizing its long-term payoff. From this understanding, the updating rule in (16) is shown to enable learning of policy

parameters. Similarly, it can be demonstrated that (24) also facilitates policy parameter learning. In fact, since

$$\nabla_{\vec{\omega}_t} \log \pi_t(s_t, a_t) = \frac{1}{\pi_t(s_t, a_t)} \nabla_{\vec{\omega}_t} \pi_t(s_t, a_t),$$

when $\delta_t > 0$, the probability of selecting action a_t will also be increased upon using (24). The main difference from (19) lies in the fact that the amount of update in (24) is further controlled by the factor $1/\pi_t(s_t, a_t)$. If $\pi_t(s_t, a_t)$ is close to 0, big changes will be applied to $\vec{\omega}_t$, in a hope of quickly increasing the chance of performing action a_t. On the other hand, if $\pi_t(s_t, a_t)$ is close to 1, the amount of update to $\vec{\omega}_t$ will be reduced. This is because action a_t is already enjoying a high selection probability, therefore learning is close to convergence. In this case, updating $\vec{\omega}_t$ slowly may be more desirable in order to stabilize the learning system. From the above discussion, it can be envisaged that (24) will help to learn policy parameters faster and more reliably, in comparison with (19). Our experiment results reported in Section VII agree with this understanding.

VII. Experiment results

The experimental study in this section focuses on maze problems, which are two-dimensional grid environments made up of three different types of positions: empty positions, obstacles, and goals. A simple example is shown in Fig. 5, where an empty position is marked by a valid system state s_i. An obstacle is denoted by T. F stands for a goal (also known as the terminal state) in the maze.

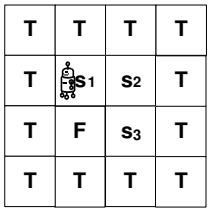

Fig. 5. An example maze problem.

At any time t, a learning agent will stay in one empty position. For example, the agent in Fig. 5 is in state s_1. It can observe its eight adjacent positions in the grid. If an adjacent position is an obstacle, the respective observation will be described by two binary bits "0" and "1". Similarly, if the adjacent position is empty, the observation will become "0" and "0". Finally, if it is a goal of the maze, the observation changes to "1" and "1". The complete observation of the agent is therefore represented through 16 binary bits. Accordingly, the condition of a classifier is also comprised of 16 attributes. Each attribute corresponds to a separate bit of an agent's observation. A condition attribute can assume one of three alternative values, namely "0", "1", and "#". Here "#" is the "*don't' care*" symbol. Following [14], in all our experiments, every condition attribute in a covering classifier is set to "#" with a probability of $P_{\#} = 0.33$.

With respect to each of its eight adjacent positions, there is a separate action that can bring the agent to that position. For example, if the agent in Fig. 5 decides to *move south*, it will end up in the goal of the maze and will therefore receive an immediate reward of 1000. Immediately after it reaches the goal, the agent will be relocated to a randomly selected empty position. Instead of moving south, if the agent decides to *move north*, it will remain in s_1 because its move will be blocked by an obstacle.

Experiments have been conducted on three groups of maze problems. First of all, to study the performance of NXCS and RXCS on traditional benchmark problems where the ultimate goal is to learn deterministic policies, experiments on Woods1, Maze5, Maze6, and Woods14 problems will be reported in Subsection VII-A. Secondly, the effectiveness of NXCS and RXCS at handling environmental randomness will be studied in Subsection VII-B by using some stochastic maze problems, including the Maze5ϵ and Maze6ϵ problems. Finally, the capability of NXCS and RXCS for coping with *perceptual aliasing* [27] will be demonstrated in Subsection VII-C with the help of the Woods101, Woods102, and Maze 7 problems. To facilitate our study, we have also utilized XCS, XCS$_\mu$ [29], XCSrwas (see Section IV), and XCS with Gradient Descent [10] as competing learning systems.

In all our experiments, for a fair comparison, we follow those parameter settings recommended in [14]. For certain parameter such as β (see (10)), a value range from 0.1 to 0.2 is considered suitable in [14]. In this case, we will use the middle value in the range. Accordingly, β is set to 0.15 in all the learning systems to be studied in this section. Another learning rate α for updating the fitness parameter is set to 0.1. Parameter ϵ_0 for identifying sufficiently accurate classifiers equals to one percent of the maximum value of the single-step reward. The discount factor γ is set to 0.71. Crossover probability χ is set to 0.75 and the mutation probability μ equals to 0.03. Additionally, threshold Θ_{GA} that controls the frequency of performing the GA component is set to 38. The threshold for deleting a classifier, Θ_{del}, is set to 20, and the subsumption threshold, Θ_{sub}, is also set to 20.

In addition to the above parameter settings, we have also tested the learning systems by setting our parameters differently. For example, the typical value for β is 0.2 in the literature [10], [29]. Instead of fixing β at 0.15, we have also examined the learning performance when $\beta = 0.2$, without noticing any significant performance difference. It is worthwhile to note that the performance of LCSs with smaller β (e.g. $\beta = 0.01$) has been studied in [10] as well. Their results suggest that reducing the value of β may help to improve the learning performance of XCS. Besides β, we have tested the learning systems when γ is set to 0.8 and 0.6. By doing that, we have observed similar outcomes as those to be presented in the following subsections.

A. Experiments on Benchmark Maze Problems

In the literature, the performance of an LCS is usually tested on several deterministic benchmark maze problems, including the Woods1, Maze5, Maze6, and Woods14 problems. All the four problems are solved by deterministic policies. They are therefore exploited to examine whether deterministic action

selection can be learned effectively through NXCS and RXCS, as we hoped. Fig. 6, Fig. 7, Fig. 8, and Fig. 9 depict the four maze problems, respectively.

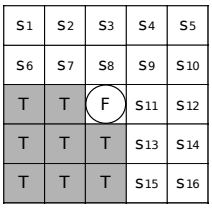

Fig. 6. Woods1 problem: each of the 16 empty positions has been labeled with a state $s_1, \ldots, s_{16}$. The goal is denoted by F. T stands for obstacles in the environment.

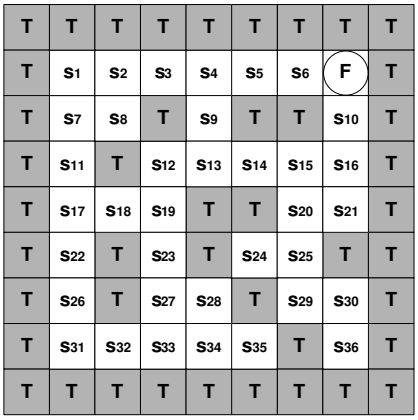

Fig. 7. Maze5 problem: each of the 36 empty positions has been labeled with a state $s_1, \ldots, s_{36}$. The goal is denoted by F. T stands for obstacles in the environment.

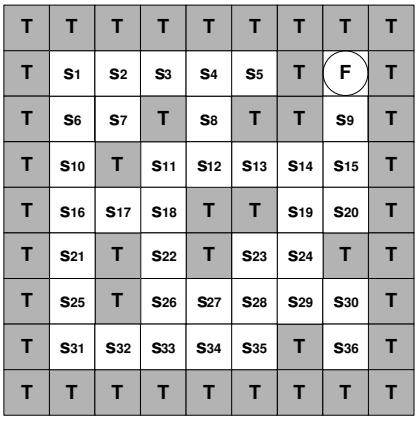

Fig. 8. Maze6 problem: each of the 36 empty positions has been labeled with a state $s_1, \ldots, s_{36}$. The goal is denoted by F. T stands for obstacles in the environment.

1) Experiments on the Woods1 problem: The Woods1 problem is perhaps the easiest Maze problem tested in this paper. It has 16 distinct states. Guided by a deterministic policy, an agent can reach the goal from any state in the grid environment through a fairly simple path, sometimes by constantly performing the same action.

The performance results from the Woods1 problem have been summarized in Fig.10. The maximum population size is

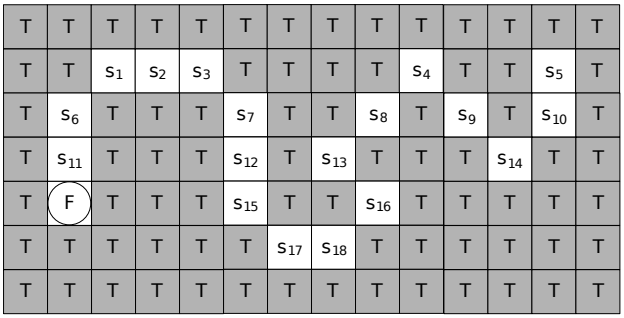

Fig. 9. Woods14 problem: each of the 18 empty positions has been labeled with a state $s_1, \ldots, s_{18}$. The goal is denoted by F. T stands for obstacles in the environment.

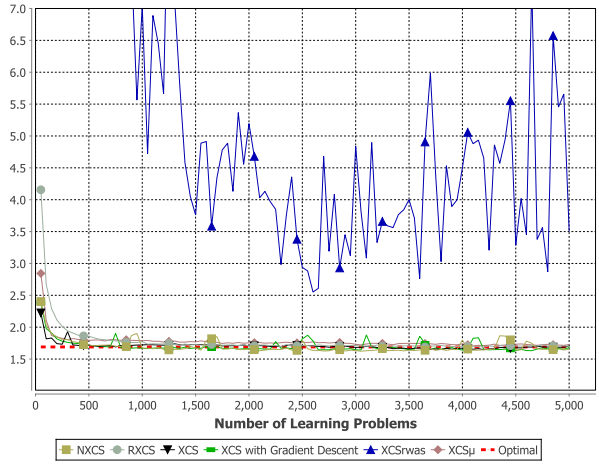

Fig. 10. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas and XCS with Gradient Descent on the Woods1 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

set to 400 classifiers in our experiments. Previous research works used similar population sizes [29]. To reduce randomness, 30 independent experiments have been conducted for each learning system. The average performance witnessed is depicted in Fig. 10. The same practice is also followed to build other result figures included in this paper.

As shown in Fig.10, after 5000 learning problems, NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas and XCS with Gradient Descent achieved average performances of 1.70, 1.70, 1.70, 1.70, 3.86, and 1.69 respectively. In comparison, the theoretical optimal performance obtainable by following the best deterministic policy is 1.69. The ANOVA test performed on the six learning systems gave rise to a p-value which is far less than 0.001. For your information, all statistical tests in this paper are performed at the final step of the learning process. Tukey's post-hoc analysis further showed that XCSrwas was outperformed by other learning systems. This can be easily seen from Fig. 10. We also found that XCSrwas was ineffective at solving other benchmark problems, including the Maze5, Maze6, and Woods14 problems. Its performance results will therefore not be further presented in this Subsection. On the other hand, Tukey's post-hoc analysis cannot find any significant performance difference among the rest of the learning

systems.

We have also measured the average time spent by each learning system for completing 5000 learning problems. In particular, the average times taken by NXCS, RXCS, and XCS are 0.639s, 0.728s, and 1.12s. Our time measurement was performed in a computer with Intel Core i5 at 3.4GHz, 8GB DDR3 memory, and Windows 7 Ultimate 64-bit operating system. Surprisingly, XCS actually requires slightly longer time than NXCS and RXCS. This might be because XCS performs on average slightly more actions in order to reach a goal in Woods1. In general, however, we do not think that the observed time difference in between any two learning systems is substantial. This observation also holds for other benchmark problems examined in this subsection.

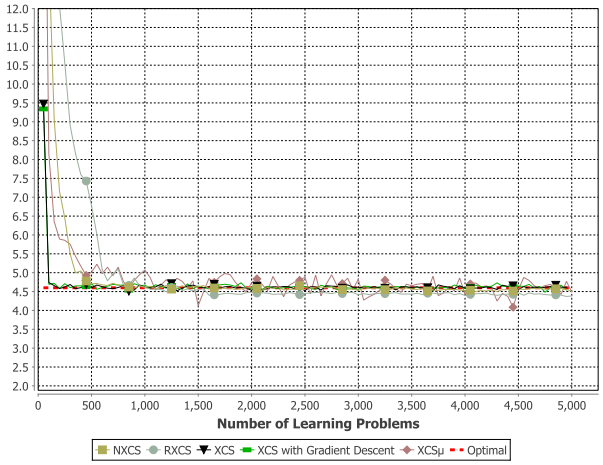

Fig. 11. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze5 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

2) Experiments on the Maze5 problem: The Maze5 problem is another benchmark multi-step reinforcement learning problem. Fig. 11 shows the performance results of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on this problem. The maximum population size equals to 3000 classifiers in the experiments. As you can see from Fig. 11, the Maze5 problem is completely solved by all the five learning systems. In particular, after 5000 learning problems, these systems eventually achieved performances of 4.61, 4.59, 4.62, 4.57, and 4.56, respectively. The theoretical optimal performance, in comparison, is 4.6. Similar with the Woods1 problem, the ANOVA test is performed and it produces a p-value of 0.93, indicating that no significant performance difference can be identified on the Maze5 problem.

3) Experiments on the Maze6 problem: Upon evaluating the performance of an LCS, the Maze6 problem is often used together with the Maze5 problem. In Fig. 12 we compare the performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze6 problem when the maximum population size is 3000 classifiers. As evident in the figure, all the five learning systems quickly approached to the theoretical optimal performance of 5.19. Specifically, after 10000 learning problems, NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent achieved average performances of 5.23, 5.38, 5.21, 5.23, and 5.18 respectively. Again the ANOVA test is performed, producing a p-value of 0.16. This result suggests that all learning systems exhibited similar performance on the Maze6 problem.

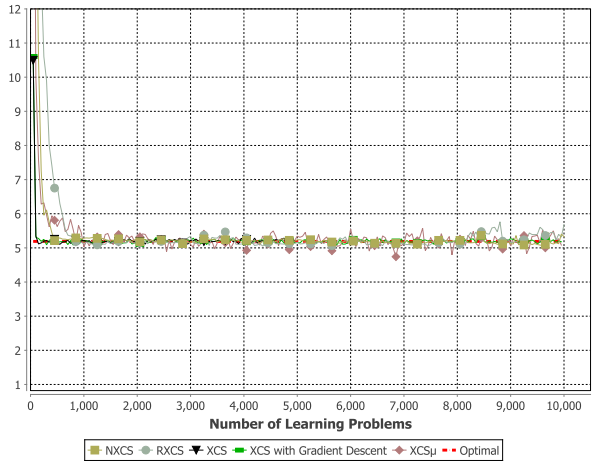

Fig. 12. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze6 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

4) Experiments on the Woods14 problem: The last set of experiments in this subsection is performed on the Woods14 problem. The Woods14 is a difficult problem. In particular, XCS has been reported as failing to solve the problem properly [10]. As depicted in Fig. 9, starting from certain state such as s_5, it will take a significant number of steps (i.e. 18) for an agent to reach the only goal located at the bottom left corner of the grid environment. Due to the long-delayed reward, it is difficult for some LCSs such as XCS to accurately approximate the state-action value function.

Fig. 13 depicts the learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Woods14 problem. Again the maximum population size is 3000 classifiers. As evident in the figure, after 20000 learning problems, NXCS and XCS with Gradient Descent obtained average performances of 9.5 and 9.9 respectively. On the other hand, the best policy in theory can achieve an average performance of 9.5. In comparison with these two learning systems, other competing learning systems appear to be less effective. To verify this observation, ANOVA test is conducted and the test gives rise to a p-value far less than 0.001. Tukey's post-hoc analysis further confirms that the observed performance difference is significant. Because NXCS and XCS with Gradient Descent behaved similarly on the Woods14 problem, a t-test is performed over the two learning systems, producing a p-value of 0.0003. This outcome suggests that NXCS may actually perform better than XCS with Gradient Descent on the Woods14 problem.

5) Evolution of a population of classifiers: Among the six learning systems that we have tested in this subsection, we believe NXCS and XCS with Gradient Descent are more effective at tackling the benchmark maze problems. In particular, both NXCS and XCS with Gradient Descent completely

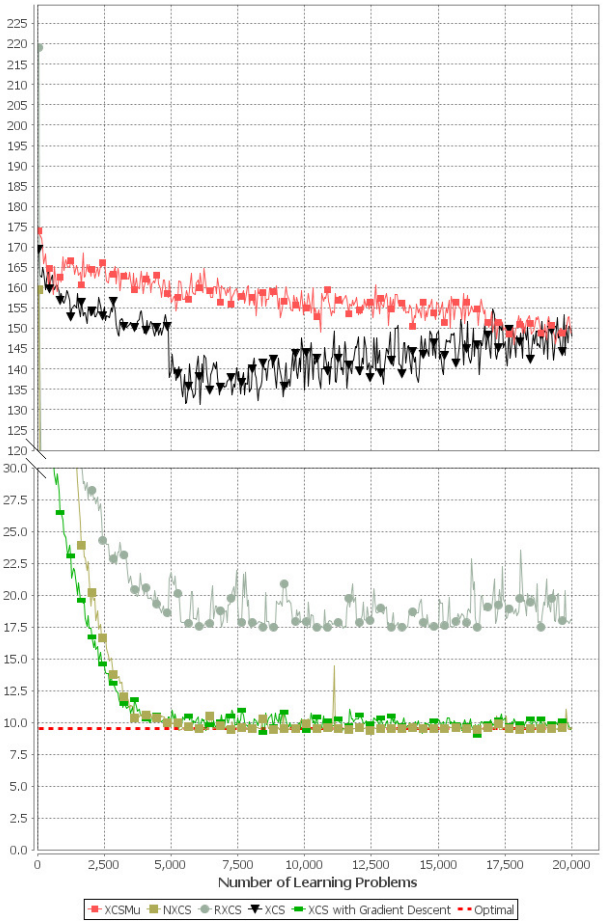

Fig. 13. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Woods14 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

solved the Woods14 problem while other learning systems failed to do so. Besides the two, RXCS seems to perform more reliably than XCS and XCS$_\mu$ on the Woods14 problem. The most ineffective system is XCSrwas. To better understand the effectiveness of NXCS and RXCS, we need to find out whether they achieved good performance by evolving a totally different population of classifiers, for example, by evolving a larger population of more specific classifiers.

Using the Woods1 problem as an example, Fig. 14 presents the *Average Classifier Specificity* of NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent during the whole learning process up to 5000 learning problems. Average classifier specificity measures the average number of non-wildcard attributes in the condition of each classifier, across the population. As illustrated in Fig. 14, for all learning systems, the level of classifier specificity will be reduced as learning continues. NXCS and XCS with Gradient Descent achieved nearly identical specificity level at the time when learning is completed. Comparatively, RXCS, XCS, and XCS$_\mu$ reached lower specificity levels. The difference with NXCS, however, is only about 2.5. We believe this is not substantial. Surprisingly, XCSrwas produced the lowest specificity level. However, since its performance is much worse than other

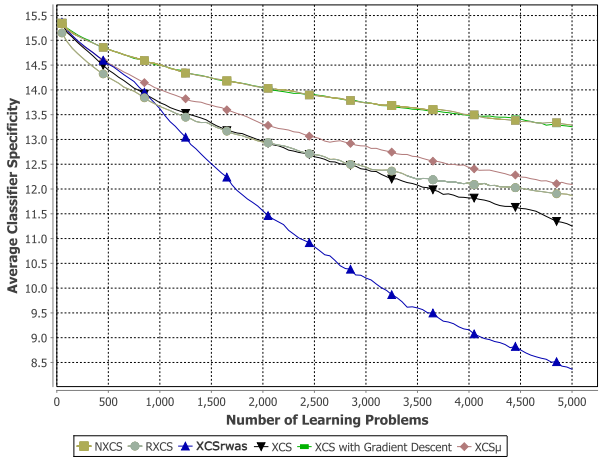

Fig. 14. Average Classifier Specificity of NXCS, RXCS, XCS and XCS$_\mu$ on the Woods1 problem. The classifier specificity is measured as the number of non-wildcard attributes in the classifier's condition.

learning systems, learning general classifiers is not its true advantage.

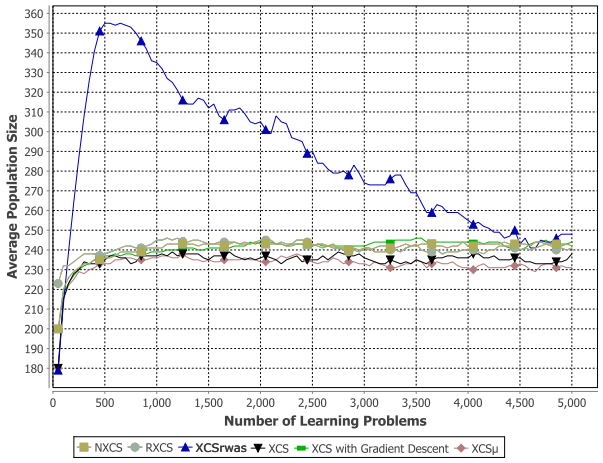

Fig. 15. The evolution of population size for NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on the Woods1 problem. Population size is measured as the number of distinct classifiers in $[P]$.

Fig.15 depicts the average population size during learning on the Woods1 problem. According to this figure, XCS and XCS$_\mu$ evolved relatively smaller population sizes than other learning systems after 5000 learning problems. Nevertheless, we do not believe the size difference is large enough to affect the practical applicability of any learning system. Fig. 14 and Fig. 15 together indicate that NXCS and RXCS will not evolve a substantially large population of more specific classifiers, in comparison with XCS, XCS$_\mu$, and XCS with Gradient Descent. This observation is again confirmed by similar patterns witnessed on other learning problems. Generally speaking, the maze problems are not particularly suitable for studying the generalization capability of LCSs, in comparison with other problems such as the multiplexer problem (which is a single-step problem). Hence, more work is needed in the future to examine how well NXCS and RXCS can generalize. Effective generalization, however, is not the key focus of this paper.

B. Experiments on Stochastic Maze Problems

The previous subsection showed that NXCS and RXCS can effectively learn deterministic policies. In this subsection, the reliability of the two learning systems in stochastic environments will be further investigated. Particularly, we have tested NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on several stochastic maze problems, including the Maze5ϵ and Maze6ϵ problems, which are stochastic extensions of the benchmark Maze5 and Maze6 problems.

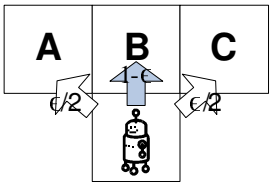

Fig. 16. Uncertainty that affects the actions performed in stochastic environments. When the agent decides to move north, with a probability of $1-\epsilon$, this action produces desirable result, relocating the agent to position B. On the other hand, with a probability of $\epsilon/2$, the agent will end up in the position A (or position C).

In both the Maze5ϵ and Maze6ϵ problems, following a common strategy utilized in [29], there is a certain degree of uncertainty, quantified through ϵ, that affects the outcome of performing any action by an agent. As illustrated in Fig. 16, after performing an action to *move north* (the intended destination is position B), the agent may end up in either position A or position C with a total probability of ϵ. Obviously the higher the value of ϵ, the higher the uncertainty in the environment. Uncertainty is common in practical applications. For example, due to flaws (or even friction) in the motors or the mechanical control system, a robot may often run away from its planned track, therefore arriving at an unexpected position. Hence, it is important that a learning system can reliably handle environmental uncertainties without losing effectiveness.

As noted in [29], when $\epsilon \geq 0.5$, XCS has difficulty of learning the optimal policy. On the other hand, XCS$_\mu$ is more robust to environmental randomness because it can clearly separate two types of prediction errors in a classifier, i.e. error due to generalization and error due to uncertainties in the environment [29]. In our experiments, we set ϵ to 0.5 in order to determine whether NXCS and RXCS will experience similar problems as XCS.

1) Experiments on the Maze5ϵ problem: The performance results for NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on the Maze5ϵ problem can be found in Fig.17. NXCS seems to perform the best over the whole learning process, developing an average performance of 8.731 after 5000 learning problems. In line with the findings reported in [29], XCS failed to converge. Instead it exhibits large fluctuations and produces an average performance of 24.036 after 5000 learning problems. XCSrwas also failed to converge, giving an average performance of 243.2 eventually. RXCS and XCS$_\mu$ appear to behave similarly, with average performances of 11.278 and 12.313 respectively at the end of the experiment. Meanwhile, XCS with Gradient Descent achieved an average performance of 11.02 after the learning process is completed.

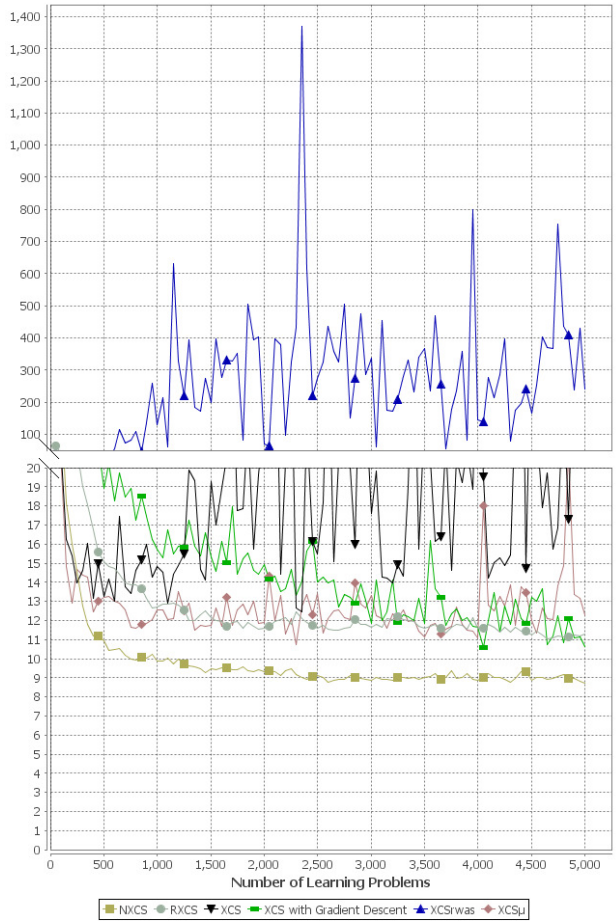

Fig. 17. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on the Maze5ϵ problem. Performance is measured as the average number of actions an agent performs in order to reach a goal.

Our ANOVA test results in a p-value far less than 0.001. Using t-test, it is subsequently confirmed that NXCS performed better than RXCS, XCS$_\mu$, and XCS with Gradient Descent. On the other hand, RXCS was not shown to perform differently from XCS$_\mu$ and XCS with Gradient Descent.

In terms of the execution time, we found that NXCS spent on average 9.23s for completing 5000 learning problems. Similarly, RXCS spent 14.6s for completing the same number of learning problems. Other learning systems tend to take similar amount of time for learning. For example, XCS$_\mu$ consumed an average of 11.3s, which is slightly higher than the time required by NXCS. We believe this is mainly because NXCS can solve the Maze5ϵ problem by performing less number of actions. In general, the difference in execution time is not substantial. NXCS and RXCS are not particularly time-consuming in comparison with other learning systems. Similar observations were also obtained for the Maze6ϵ problem.

2) Experiments on the Maze6ϵ problem: Fig.18 shows the performance results of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze6ϵ problem. Evidently, NXCS, RXCS and XCS$_\mu$ can manage to stabilize their performance eventually, reaching average performances of 9.886, 14.168 and 19.184 respectively after 10000 learning problems. Conversely, XCS and XCS with Gradient Descent failed to

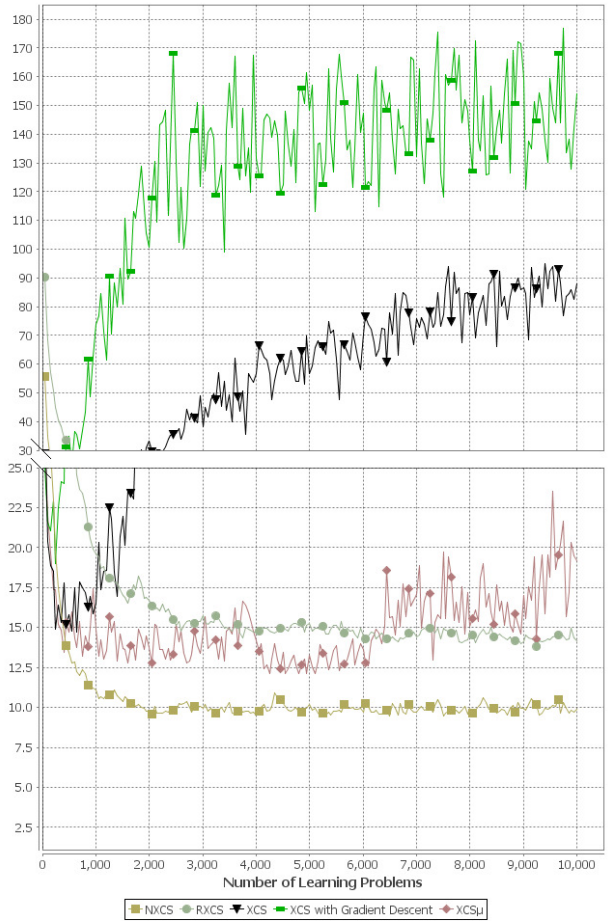

Fig. 18. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze6ϵ problem. Performance is measured as the average number of actions an agent performs in order to reach a goal.

converge properly. The observed performance difference is supported by the ANOVA test, which leads to a p-value far less than 0.001. Meanwhile, by using t-test, the performance of NXCS is shown to be significantly better than that of RXCS and XCS$_\mu$. Similarly, t-test result also suggests that RXCS performed better than XCS$_\mu$, after completing 10000 learning problems.

Besides the Maze5ϵ and Maze6ϵ problems, we have also tested our learning systems on other stochastic problems such as the Maze4ϵ problem. Similarly results were observed but the details will not be presented here. The success of NXCS and RXCS on stochastic problems shows that environmental randomness can be effectively tackled by using appropriate gradient descent methods, in particular the natural gradient learning technique studied in this paper.

C. Experiments on Partially Observable Maze Problems

Learning stochastic polices is not only beneficial at handling environmental randomness, but also desirable for coping with observational limitations in a learning environment. In this subsection, we will study this issue by performing experiments on the Woods101, Woods102 and Maze7 problems.

1) Experiments on the Woods101 problem: The Woods101 problem, as depicted in Fig. 19, is a small grid environment that consists of 10 empty positions. It is interesting to note that, based on an agent's perception in the Woods101 problem, two states i.e. s_2 and s_4, are indistinguishable. Whenever a deterministic policy is followed, the same action will be performed in both states s_2 and s_4. There is hence a chance for the agent to be trapped in a local loop. In comparison, if a stochastic policy is utilized, the agent may choose to perform different actions in states s_2 and s_4, increasing the chance of escaping from the loop.

T	T	T	T	T	T	T
T	S1	S2	S3	S4	S5	T
T	S6	T	S7	T	S8	T
T	S9	T	F	T	S10	T
T	T	T	T	T	T	T

Fig. 19. The Woods101 problem: each of the 10 empty positions has been labeled with a state $s_1, \ldots, s_{10}$. The goal is denoted by F. T stands for obstacles in the environment.

Due to the use of the greedy action selection strategy while evaluating XCS, XCS$_\mu$, and XCS with Gradient Descent, if the agent is trapped in a local loop, evaluation will continue from a randomly selected new state. Without closely tracking its move in the environment, a loop is simply identified whenever the agent cannot reach a goal after 50 consecutive moves.

Fig. 20 presents the performance of NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on the Woods101 problem. As can be seen from this figure, NXCS and RXCS appear to perform much better than the other four learning systems throughout the whole learning process. In particular, at the end of the experiment (i.e. 8000 learning problems), the average performances achieved by NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent are 4.32, 4.33, 63.09, 40.35, 54.60, and 62.6 respectively. This observation agrees well with the outcome of the ANOVA test which produces a p-value far less than 0.001. Through t-tests, we have further identified the learning systems that perform differently. For example, the p-value is far less than 0.001 for the t-test between NXCS and XCS or between NXCS and XCS with Gradient Descent. On the other hand, the performance of NXCS and RXCS shows little difference.

Besides the above results, we also found that, after about 2000 learning problems, NXCS and RXCS achieved an average performance that is very close to the theoretical optimum of 4.3. This optimum is obtained by theoretically evaluating the performance of the best stochastic policy (determined manually) without any memory component.

2) Experiments on the Woods102 problem: The Woods102 problem, as depicted in Fig. 21, is another partially observable maze problem. There are four states, i.e. s_7, s_9, s_{18}, and s_{20}, with identical observations in this problem. Fig. 22 shows the performance results on the Woods102 problem. Similar with the Woods101 problem, it is apparent that NXCS and

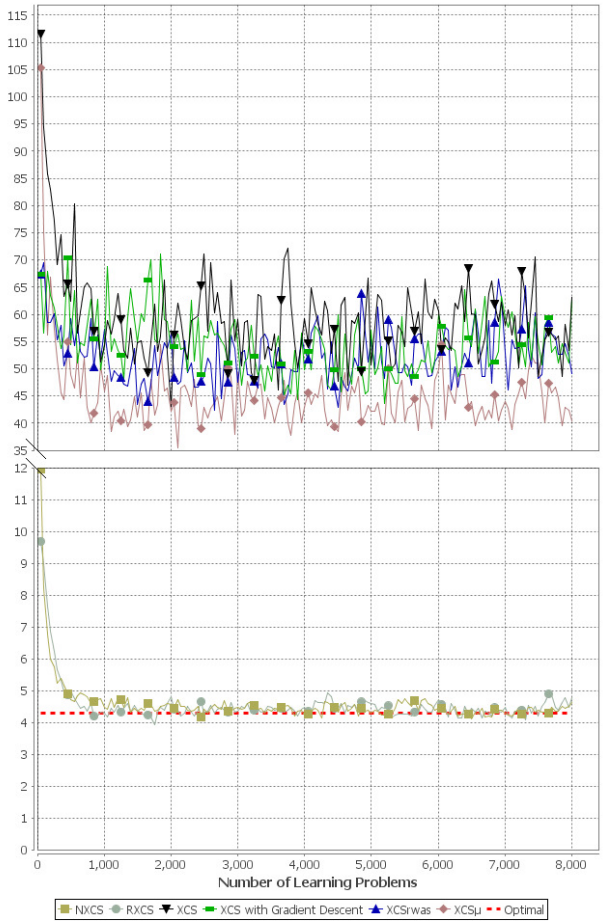

Fig. 20. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent on the Woods101 problem. The performance is measured as the average number of actions to be performed by an agent in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

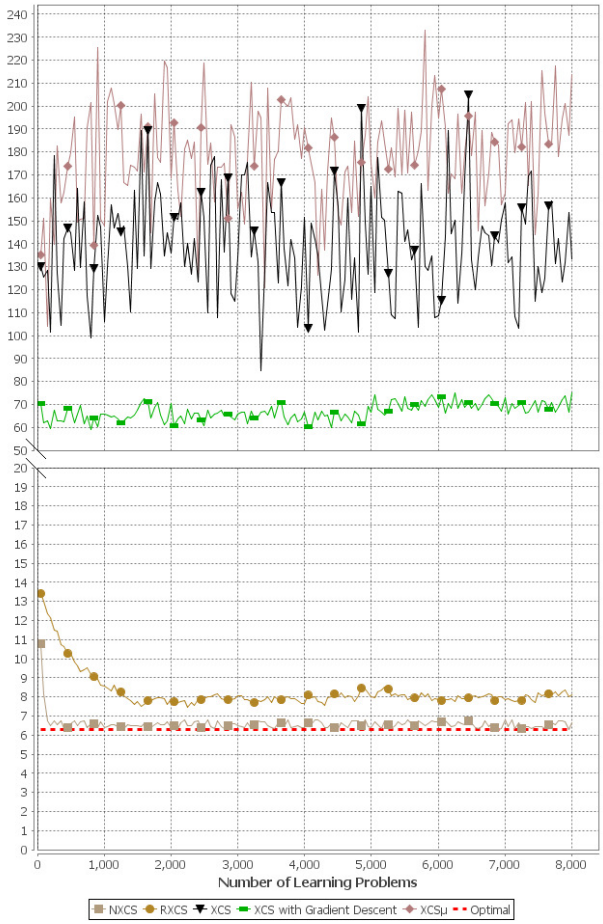

Fig. 22. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Woods102 problem. The performance is measured as the average number of actions to be performed by an agent in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

T	T	T	T	T	T	T
T	S_1	T	F	T	S_2	T
T	S_3	T	S_4	T	S_5	T
T	S_6	S_7	S_8	S_9	S_{10}	T
T	S_{11}	T	S_{12}	T	S_{13}	T
T	T	T	T	T	T	T
T	S_{14}	T	S_{15}	T	S_{16}	T
T	S_{17}	S_{18}	S_{19}	S_{20}	S_{21}	T
T	S_{22}	T	S_{23}	T	S_{24}	T
T	S_{25}	T	F	T	S_{26}	T
T	T	T	T	T	T	T

Fig. 21. Woods102 problem: each of the 26 empty positions has been labeled with a state $s_1, \ldots, s_{26}$. The goal is denoted by F. T stands for obstacles in the environment.

RXCS performed much better than XCS, XCS$_\mu$, and XCS with Gradient Descent (the ANOVA test again produced a p-value far less than 0.001). Meanwhile, XCSrwas performed poorly but the corresponding result was not depicted in Fig. 22.

We also found that NXCS and RXCS converged to average performances of 6.57 and 7.81 respectively, after 8000 learning problems. The t-test subsequently shows that the performance difference between NXCS and RXCS is significant (p-value < 0.001). Particularly, NXCS achieved the best performance among all learning systems. Its average performance is the closest to the theoretical optimal of 6.15, with a small difference of only 0.42.

3) Experiments on the Maze7 problem: In comparison with the Woods101 and Woods102 problems, the Maze7 problem, as depicted in Fig. 23, is more difficult because the agent may need to go through two aliasing states, namely s_6 and s_7, before reaching a goal. Moreover, one aliasing state, i.e. s_7, is very far from the goal.

We can see the performance results of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze7 problem in Fig.24. The results clearly showed that NXCS and RXCS performed better than XCS, XCS$_\mu$, and XCS with Gradient Descent over the entire learning process (the p-value of the ANOVA test is far less than 0.001). NXCS and RXCS achieved

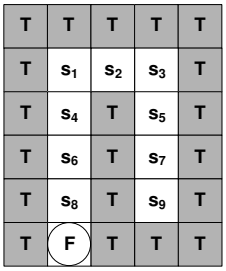

Fig. 23. Maze7 problem: each of the 9 empty positions has been labeled with a state $s_1, \ldots, s_9$. The goal is denoted by F. T stands for obstacles in the environment.

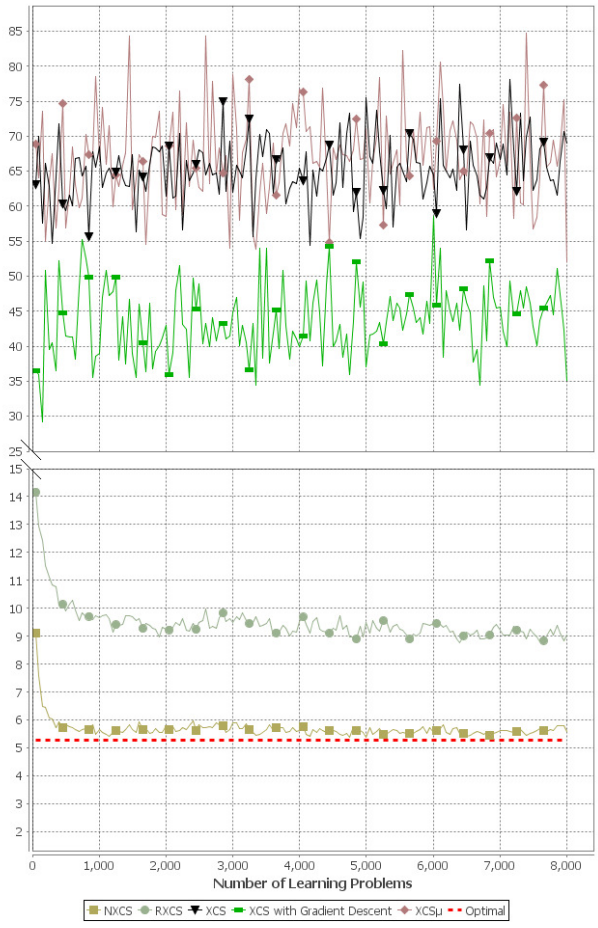

Fig. 24. Learning performance of NXCS, RXCS, XCS, XCS$_\mu$, and XCS with Gradient Descent on the Maze7 problem. Performance is measured as the average number of actions an agent performs in order to reach a goal. The theoretical optimal performance is also indicated in this figure.

average performances of 6.02 and 9.05 respectively after 8000 learning problems. The t-test between NXCS and RXCS produced a p-value far less than 0.001, confirming that even NXCS and RXCS exhibit distinguishable performances. In particular, the performance of NXCS is closer to the theoretical optimal of 5.88, with a slight difference of 0.14.

4) Example learned classifiers: The experiment results presented in this subsection clearly showed that, by learning stochastic policies, NXCS and RXCS can more effectively handle partially observable environments than XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent. NXCS is also shown to perform better than RXCS, thanks to its use of a natural gradient learning technique.

Table I presents some example high-fitness classifiers learned by NXCS from the Woods101 problem. All these classifiers have reasonably long experience and high fitness (i.e. close to 1). Classifier C1 in Table I matches with state s_7 in Fig. 19. Both C2 and C3 match with states s_2 and s_4. C4 matches with s_6. Finally C5 matches with both s_9 and s_{10}.

TABLE I
SELECTED HIGH-FITNESS CLASSIFIERS LEARNED BY NXCS FROM THE WOODS101 PROBLEM. THE MAXIMUM POPULATION SIZE IS 1000. A TOTAL OF 246 CLASSIFIERS HAVE BEEN EVOLVED IN THE POPULATION.

ID	Condition	Action	Prediction	Numero-sity	Fitness	Experience	theta
C1	00***001*101*101	SOUTH	1000.00	294	0.82	7891.00	8.70
C2	0*0**1**0*0001**	SOUTH_WEST	863.52	57	0.84	7916.00	1.90
C3	01*10100*0*0**0*	SOUTH_EAST	575.54	93	0.75	2547.00	1.65
C4	0100000***01*00*	NORTH_EAST	636.22	173	0.74	5387.00	2.45
C5	**0*010*01010101	NORTH	477.94	34	0.84	778.00	1.54

As recommended by C1, if an agent moves south from s_7, it will reach a goal and receive an immediate reward of 1000.0. As a result, the prediction of C1 is 1000.0. To ensure a high probability for the agent to move south, the policy parameter ω in C1 assumes a high value of 8.70, agreeing with our expectation. As we already mentioned, states s_2 and s_4 in Fig. 19 share identical observations. When an agent is in either s_2 or s_4, the optimal decision is to either move southwest or move southeast with equal probability. As witnessed in Table I, C2 gives the recommendation to move southwest and C3 suggests to move southeast instead. Based on the policy parameters of C2 and C3 alone, it can be calculated that the probability of moving southwest is 0.56 and the probability of moving southeast is 0.44. We found that this difference in probability does not lead to serious deterioration in performance since an agent can still easily escape from a local loop. Finally, we note that, regardless of whether the agent is in state s_9 or s_{10}, the best decision is to move north. Therefore C5 serves as a good generalization over the two states and it makes the correct recommendation.

VIII. CONCLUSIONS

This paper studied a new method for learning stochastic policies in LCSs based on a policy gradient framework. Our research was motivated by the understanding that many existing LCSs were designed to solve reinforcement learning problems by learning state-action value functions and by using a fixed action selection strategy such as the greedy strategy. In this paper, through adaptive learning of action selection

strategies, we have the goal to improve the performance of LCSs, especially when the learning environment is stochastic and partially observable.

Using XCS as the basis, two new learning systems have been developed successfully in this paper. One system, termed the RXCS, was designed to learn policy parameters by following a regular gradient learning approach. The other system, termed the NXCS, was constructed based on a new natural gradient learning method. To the best of our knowledge, our research presented the first study of learning stochastic policies in LCSs by using both the regular gradient and natural gradient technologies.

Experiments have been conducted on three groups of maze problems. The results clearly showed the advantage of learning stochastic policies. In comparison with XCS, XCS$_\mu$, XCSrwas, and XCS with Gradient Descent, NXCS and RXCS are clearly more robust to state aliasing and environmental randomness. They also performed competitively on traditional benchmark problems that are solved by learning deterministic policies. Through analyzing the learning process and some learned classifiers (see Table I), it is further confirmed that the performance gain achieved by NXCS and RXCS did not come at a price of evolving a larger population of more specific classifiers.

Looking into the future, we would hope to see interesting applications of NXCS and RXCS in real-world systems that require sequential and stochastic decision making. The usefulness of NXCS and RXCS for a wide range of machine learning problems, including data mining and pattern recognition tasks, may also deserve in-depth investigation. Meanwhile, it is curious to see whether substantial theoretical analysis of NXCS and RXCS can be performed to deepen our understanding of their asymptotic behaviors including convergence properties. Finally, we note that, in addition to XCS, the regular and natural gradient learning technologies may potentially work well with other types of LCSs. More efforts are therefore necessary to fully develop LCS as an effective platform for learning and supporting stochastic decision making.

APPENDIX A
LEARNING POLICY PARAMETERS THROUGH NATURAL GRADIENT

This appendix explains how to derive the updating rule in (24) for learning policy parameters based on natural gradient. As has been proven in [46], the derivative of J with respect to $\vec{\omega}_t$ can be computed from

$$\nabla_{\vec{\omega}_t} J = \sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \nabla_{\vec{\omega}_t} \pi_t(s,a) \cdot Q_{\pi_t}(s,a) \tag{25}$$

The appearance of Q_{π_t} in (25) suggests that we must approximate Q_{π_t} first before estimating $\nabla_{\vec{\omega}_t} J$. Similar with [46], let $\vec{w}_t$ be a vector of N dimensions (N is also the dimension of $\vec{\omega}$), then an approximation of Q_{π_t} as given below

$$Q_{\pi_t}(s,a) \approx \vec{w}_t^T \cdot \nabla_{\vec{\omega}_t} \log \pi_t(s,a) \tag{26}$$

is called *compatible* with the policy parameterization. Accordingly the vector $\vec{\psi}_{s,a} = \nabla_{\vec{\omega}_t} \log \pi_t(s,a)$ will be called the *compatible action-state vector*. Provided that $\vec{w}_t^*$ is the vector that reduces the approximation error in (26) to its minimum, then the following can be easily shown to be correct [46]:

$$\nabla_{\vec{\omega}_t} J = \sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \nabla_{\vec{\omega}_t} \pi_t(s,a) \cdot \left(\vec{\psi}_{s,a}^T \cdot \vec{w}_t^*\right) \tag{27}$$

In particular, let us define the approximation error as

$$\varepsilon(\vec{w}_t) = \sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \pi_t(s,a) \left(Q_{\pi_t}(s,a) - \vec{\psi}_{s,a}^T \cdot \vec{w}_t\right)^2$$

Using the fact that $\nabla_{\vec{w}_t^*} \varepsilon = 0$, we have

$$\begin{aligned} \nabla_{\vec{w}_t^*} \varepsilon = & -2 \cdot \sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \pi_t(s,a) \left(\begin{array}{l} Q_{\pi_t}(s,a) - \\ \vec{\psi}_{s,a}^T \cdot \vec{w}_t^* \end{array} \right) \cdot \vec{\psi}_{s,a} \\ = & \ 0 \end{aligned} \tag{28}$$

Since

$$\sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \pi_t(s,a) Q_{\pi_t}(s,a) \cdot \vec{\psi}_{s,a} = \nabla_{\vec{\omega}_t} J$$

From (28), we further have

$$\nabla_{\vec{\omega}_t} J = \left(\sum_{s\in S} d^{\pi_t}(s) \sum_{a\in A} \pi_t(s,a) \vec{\psi}_{s,a} \cdot \vec{\psi}_{s,a}^T \right) \cdot \vec{w}_t^* \tag{29}$$

Based on (22) and (23), it can be immediately seen that

$$\vec{w}_t^* = \tilde{\nabla}_{\vec{\omega}_t} J \tag{30}$$

Up to this point it becomes clear that, in order to update $\vec{\omega}_t$ according to its natural gradient, all we need is to find $\vec{w}_t^*$. It is to be noted here that, instead of using $\vec{\psi}_{s,a}^T \cdot \vec{w}_t$ to approximate $Q_{\pi_t}(s,a)$, the literature shows that it is a better idea to approximate the *advantage function* defined in (14) [6], [25]. Accordingly, by replacing Q_{π_t} in (28) with A_{π_t}, it is clear that

$$\nabla_{\vec{w}_t} \varepsilon(\vec{w}_t) \approx -2 \cdot \left(\delta_t - \vec{\psi}_{s,a}^T \cdot \vec{w}_t\right) \cdot \vec{\psi}_{s,a} \tag{31}$$

Remember that δ_t in (31) is defined in (16). It serves as a local approximation of the advantage function. Notice that by reducing $\varepsilon(\vec{w}_t)$ to its minimum, $\vec{w}_t^*$, which is the natural gradient of J at time t, can be obtained. Consequently, to approximate $\vec{w}_t^*$, we should update $\vec{w}_t$ along the direction given in (31). A learning rule for $\vec{w}_t$ therefore is

$$\vec{w}_{t+1} \leftarrow \vec{w}_t - \lambda \cdot \left(\vec{\psi}_{s,a} \cdot \vec{\psi}_{s,a}^T \cdot \vec{w}_t - \delta_t \cdot \vec{\psi}_{s,a}\right) \tag{32}$$

Based on (32), given $\vec{w}$ as an initial approximation of $\tilde{\nabla}_{\vec{\omega}_t} J$, it becomes immediate that

$$\tilde{\nabla}_{\vec{\omega}_t} J \ \tilde{\propto}\ \vec{w} + \lambda \cdot \delta_t \cdot \vec{\psi}_{s,a} - \lambda \cdot \vec{\psi}_{s,a} \cdot \vec{\psi}_{s,a}^T \cdot \vec{w}$$

Using the above, we can finally derive the updating rule for $\vec{\omega}_t$, as shown in (24).

ACKNOWLEDGMENT

This work was supported in part by the Research Establishment Grant 204636/3255 and the University Research Fund under Grant 206971/3468 from the Victoria University of Wellington, and in part by the Marsden Fund of New Zealand Government under Contract VUW1209.

REFERENCES

[1] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[2] J. Bacardit, E. Bernadò-Mansill, and M. V. Butz, "Learning classifier systems: Looking back and glimpsing ahead," in *Learning Classifier Systems - Lecture Notes in Computer Science Volume 4998*. Springer Berlin Heidelberg, 2008, pp. 1–21.

[3] A. M. Barry, J. H. Homes, and X. Llorà, "Data mining using learning classifier systems," in *Applications of Learning Classifier Systems, Studies in Fuzziness and Soft Computing*, L. Bull, Ed. Springer-Verlag, 2004, pp. 15–67.

[4] J. Beitelspacher, J. Fager, G. Henriques, and A. McGovern, "Policy gradient vs. value function approximation: A reinforcement learning shootout," School of Computer Science, University of Oklahoma, Tech. Rep., 2006.

[5] E. Bernadó, X. Llorà, and J. M. Garrell, "A comparative study of two learning classifier systems on data mining," in *Advances in Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Springer-Verlag, 2002, pp. 115–132.

[6] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Journal Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[7] A. Budd, C. Stone, J. Masere, A. Adamatzky, B. DeLacyCostello, and L. Bull, "Towards machine learning control of chemical computers," in *From utopian to genuine unconventional computers*, A. A. A and C. Teuscher, Eds. Luniver Press, 2006.

[8] L. Bull and T. O'Hara, "Accuracy-based neuro and neuro-fuzzy classifier systems," in *GECCO 2002: Proceedings of the genetic and evolutionary computation conference*, 2002, pp. 905–911.

[9] M. V. Butz, "Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005, pp. 1835–1842.

[10] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems," *IEEE Transactions on Evolutionary Computation*, 2005.

[11] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 28–46, 2004.

[12] M. V. Butz, P. L. Lanzi, and S. W. Wilson, "Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, pp. 355–376, 2008.

[13] M. V. Butz and P. O. Stalph, "Modularization of XCSF for multiple output dimensions," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 1243–1250.

[14] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," in *Advances in Learning Classifier Systems, Lecture Notes in Computer Science Volume 1996*. Springer Berlin Heidelberg, 2001, pp. 253–272.

[15] Y. Cao, N. Ireson, L. Bull, and R. Miles, "Design of a traffic junction controller using a classifier system and fuzzy logic," in *Proceedings of the sixth international conference on computa- tional intelligence, theory, and applications*, 1999.

[16] J. Casillas, B. Carse, and L. Bull, "Fuzzy-XCS: A michigan genetic fuzzy system," *IEEE Transactions on Fuzzy Systems*, pp. 536–550, 2007.

[17] J. Casillas, B. Carse, and L. Bull, "Reinforcement learning by an accuracy-based fuzzy classifier system with real-valued output," in *International Workshop on Genetic Fuzzy Systems*, 2008.

[18] G. Chen, Z. H. Yang, H. He, and K. M. Goh, "Coordinating multiple agents via reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 10, no. 3, pp. 273–328, 2005.

[19] G. Chen, M. Zhang, S. Pang, and C. Douch, "Stochastic decision making in learning classifier systems through a natural policy gradient method," in *Proceedings of 21st International Conference on Neural Information Processing*. Springer, 2014.

[20] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, 2013.

[21] D. Gu and H. Hu, "Accuracy based fuzzy q-learning for robot behaviours," in *Proceedings of 2004 IEEE International Conference on Fuzzy Systems*, 2004.

[22] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[23] J. H. Holland, "Adaptation," in *Progress in theoretical biology*, R. Rosen and F. Snell, Eds. Academic Press, 1976, pp. 263–293.

[24] A. J. Ijspeert and S. Schaal, "Learning Attractor Landscapes for Learning Motor Primitives," in *Advances in Neural Information Processing Systems 15, (NIPS 2003)*. MIT Press, 2003, pp. 1523–1530.

[25] S. Kakade, "A natural policy gradient," 2001.

[26] N. Kohl and P. Stone, "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion," in *Proceedings of the IEEE International Conference for Robotics and Automation (ICRA)*. IEEE Press, 2003.

[27] P. L. Lanzi, "An analysis of the memory mechanism of XCSM," in *Proceedings of the Third Genetic Programming Conference*, 1998, pp. 643–651.

[28] P. L. Lanzi, "Learning classifier systems: then and now," *Evolutionary Intelligence*, 2008.

[29] P. L. Lanzi and M. Colombetti, "An extension to the XCS classifier system for stochastic environments," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 353–360.

[30] P. L. Lanzi and D. Loiacono, "Classifier systems that compute action mappings," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 1822–1829.

[31] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "XCS with computed prediction in continuous multistep environments," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, 2005, pp. 2032–2039.

[32] P. L. Lanzi and S. W. Wilson, "Toward optimal classifier system performance in nonmarkov environments," *Evolutionary Computation Journal*, vol. 8, pp. 393–418, 2000.

[33] D. Loiacono and P. L. Lanzi, "Evolving neural networks for classifier prediction with XCSF," in *Proceedings of the ECAI'06 Workshop on Evolutionary Computation*, 2006, pp. 36–40.

[34] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 11, no. 5, 2010.

[35] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS)*. IEEE Press, 2006.

[36] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, pp. 1180–1190, 2008.

[37] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of Third IEEE-RAS International Conference on Humanoid Robots,*. IEEE Press, 2003.

[38] R. Preen and L. Bull, "Discrete dynamical genetic programming in XCS," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 1299–1306.

[39] K. Shafi, H. A. Abbass, and W. Zhu, "The role of early stopping and population size in XCS for intrusion detection," in *SEAL, Lecture notes in computer science*, T. D. Wang, X. Li, S. H. Chen, X. Wang, H. A. Abbass, H. Iba, G. Chen, and X. Yao, Eds. Springer, 2006, pp. 50–57.

[40] F. Shoeleh, A. Hamzeh, and S. Hashemi, "Towards final rule set reduction in XCS: a fuzzy representation approach," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 1211–1218.

[41] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Learning without state-estimation in partially observable markovian decision processes," in *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994, pp. 284–292.

[42] P. O. Stalph, M. V. Butz, and G. K. M. Pedersen, "Controlling a four degree of freedom arm in 3d using the XCSF learning classifier system," in *Proceedings of the 32nd Annual German Conference on AI*, 2009, pp. 193–200.

[43] M. Studley, "Learning classifier systems for multi-objective robot control," Ph.D. dissertation, Faculty of Computing, Engineer- ing and Mathematics University of the West of England, 2006.

[44] M. Studley and L. Bull, "X-TCS: accuracy-based learning classifier system robotics," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005, pp. 2099–2106.

[45] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[46] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12 (NIPS 1999)*. MIT Press, 2000, pp. 1057–1063.

[47] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, 2009.

[48] R. J. Urbanowicz and J. H. Moore, "The application of michigan-style learning classifier systems to address genetic heterogeneity and epistasisin association studies," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010, pp. 195–202.

[49] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[50] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.