Learning Classifier System on a Humanoid NAO Robot in Dynamic Environments

Chang Wang, Pascal Wiggers, Koen Hindriks, Catholijn M. Jonker
Interactive Intelligence Section, Department of Intelligent Systems, Faculty of EEMCS
Delft University of Technology, Mekelweg 4, 2628CD Delft, the Netherlands
Email: c.wang-2@tudelft.nl

***Abstract*—We present a modified version of Extended Classifier System (XCS) on a humanoid NAO robot. The robot is capable of learning a complete, accurate, and maximally general map of an environment through evolutionary search and reinforcement learning. The standard alternation between explore and exploit trials is revised so that the robot relearns only when necessary. This modification makes the learning more effective and provides the XCS with external memory to evaluate the environmental change. Furthermore, it overcomes the drawbacks of learning rate settings in traditional XCS. A simple object seeking task is presented which demonstrates the desirable adaptivity of LCS for a sequential task on a real robot in dynamic environments.**

I. Introduction

A Learning Classifier System (LCS) is a genetic algorithms based machine learning paradigm that usually combines temporal difference [1] or supervised learning [2] with a genetic algorithm [3] to solve problems [4]. Originally, learning in LCS is modeled as an on-line process to adapt to an unknown environment, and the solution to the target problem is represented as a population of condition-action-prediction rules, maintained by the classifier system which tries to maximize the actual reward from the environment by taking the selected actions under the given conditions. Such a paradigm enables incremental and on-line learning in dynamic environments where a real robot should know the environmental changes and react accordingly rather than to perform only preprogrammed behaviors in static environments. In comparison to a purely evolutionary approach, an LCS is faster to learn due to the reinforcement learning techniques. In addition, its inherent ability to generalize over the state-action space and its explicit representation of the learned rules make the robots' behaviors easier to understand for human operators.

However, there are not many implementations of LCS on real robots. Some applications are on simple robots with only sonar or light sensors so that the robotic tasks are quite limited and time consuming, which typically take several hours and hundred of trials. This paper describes an application of the popular accuracy-based LCS, XCS, on a widely used advanced humanoid Nao robot that enables fast robot learning and efficient human-robot interaction. The main contribution of this paper is that we tailor the standard explore/exploit learning framework of XCS for robotic tasks and propose to use extra memory for the goal states achieved in exploit trials. In addition, the prediction learning rate is found to be the key factor that affects the relearning speed in a new environment. The paper is organized as follows: Section II introduces Learning Classifier Systems and related applications on robots. Section III gives a detailed description of our classifier system implementation for a humanoid Nao robot. Section IV describes our robot experiments and analyzes the results. Section V concludes the paper and discusses the future work.

II. Related Work

The original Learning Classifier System was invented by Holland as a cognitive model [5]. However, there was hardly a real implementation of LCS until Wilson introduced a simplified version, ZCS, which keeps much of Hollands original framework and has close relations to Q-learning [6]. There followed the successful extended classifier system (XCS) [7], an accuracy-based classifier system that differs from earlier strength-based approaches in the way that it calculates the accuracy of rules instead of the payoff. Detailed comparisons of these two approaches are given in [8].

When applying LCS to real robotic tasks, more problems need to be considered than in simulation environments. The most important one is the time limit. Several thousands of trials are hardly available for real robotic experiments [9]. The Interactive Classifier System (ICS) describes a fast learning method for a mobile robot which acquires autonomous behaviors from interaction between a human and a robot [10]. The influence of time can be considered in the reward functions [11]. Other inspiring applications of LCS on real robots include the robot-shaping experiment [12], Fuzzy Classifier System (FCS) [13], Neural Classifier System (NCS) [14], X-TCS [15], enhanced LCS [16], and non-reactive LCS [17].

However, a large number of parameters in LCS are responsible for its performance. Therefore, parameter setting has always been a big problem in LCS applications. For example, the fitness sharing parameters need to be adjusted correctly to enable ZCS perform optimally in simple multi-step maze problems [18]. In comparison, XCS and X-TCS don't require such careful tuning of parameters to achieve optimal behavior [11]. But improvements are necessary when XCS works in noisy and dynamic rather than static environments where separate learning rates are suggested for prediction update, error update and fitness update in solving one-step *multiplexer* problem with XCS [19].

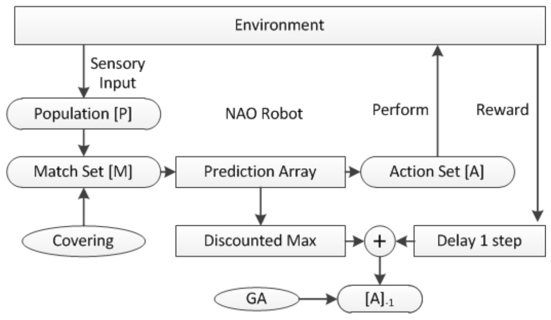


Fig. 1. An overview of XCS

III. Learning Classifier System on NAO

A. The Platform

In our experiments, we used the academic version of humanoid NAO robots. It is 58 cm tall and has an AMD Geode 500 MHz CPU, 256 MB SDRAM and an Embedded Linux system. Ethernet or WiFi connections to computers are both available. The forehead camera is used as the main input source to the classifier system, and the movement of NAO's head is controlled by the classifier system which sends Urbi scripts to the NAO with default APIs. In addition, the speakers on its head enable text-to-speech function. Additional sensors for further usage include microphones, gyrometers, accelerometer, sonars, bumpers and also kinect.

A client-server architecture is chosen to control the NAO remotely from a GUI on a computer. The sensory inputs of the NAO are sent to the computer where images are processed and action commands are given. The XCSlib [20] is modified into the LCS controller here, which uses evolutionary search and reinforcement learning to evolve complete, accurate, and maximally general payoff map of an environment. A detailed algorithmic description is given as follows.

B. An Overview of LCS on NAO

The XCS is modified in several ways to apply on NAO robot. First, we propose to stop exploration when the NAO has learned an optimal policy and continue with exploitation until the environment change is detected. Second, consistent actions are used in explore trials to speed up the learning process while in exploit trials, the system remembers the goal state achieved and the number of steps towards the goal state. In addition, separate learning rates are used for prediction update, error update and fitness update rather than the same constant value in traditional XCS experiments. These features are explained in Section III-D and Section III-E. An overview of XCS used on NAO is given in Fig.1 [8].

C. Knowledge Representation

Generally, the current knowledge of the classifier system is represented as a population [P] of N classifiers that are basically condition-action rules. If N is fixed and this limit is reached, old classifiers are deleted to make room for new ones. At the first time step, [P] can be initialized empty and new classifiers are created by a scheme called *covering* [21] which guarantees the current input be matched by at least one classifier's condition part. Besides the condition and action parts, each classifier in XCS maintains three important statistics: p, the predicted reward of the classifier's action; ϵ, an estimate of the prediction's error; F, the fitness of the classifier used for rule discovery.

The traditional ternary language uses $\{0,1,\#\}$ for the condition part and binary strings for the action part. Prediction is usually initialized as described in [22]. Actually, the representation depends on the chosen task. For example, in Wilson's *Woods* examples that a simulated robot moves in a maze searching for food, a 3 bits binary string is preferred to encode the 8 moving directions, while in TCS's light-seeking example, an 'Unordered Bound Representation' is better for the continuous sonar sensory inputs [23]. However, when the input of the LCS is an image rather than proximity sensors, it needs feature extraction to decide the length and value of the input string. In our case, each input image is simply segmented into 9 parts, each part thresholds the target pixels and a standard binary representation is used.

D. Performance Component

The performance component controls the system's behavior. In Wilson's XCS, either a pure explore trial or a pure exploit trial is performed on each time step. Usually, the system alternates between the two. In other words, learning happens only in explore mode while evaluation happens in exploit mode.

1) Explore Trials: A random action is usually selected from the rules in the match set [M] to match the current sensory input. However, random actions are inefficient for robots, e.g., moving back and forwards without changing its current state. Therefore, consistent actions are adopted to avoid learning in situations where the state of the robot remains the same after one action. In other words, LCS matches an input to create a match set [M] and an action set [A], then performs the selected action until there is a significant change in sensory input. If the new input still matches all the classifiers in [A], then it continues with the current action. Else if none of the classifiers in [A] matches the current input, [A] is deleted. Otherwise, a probabilistic selection takes place among them.

2) Exploit Trials: The system deterministically selects the highly recommended action with the highest prediction so that an optimal action sequence is chosen to achieve the goal state. However, if the environment changes, the system needs to relearn the optimal policy. Traditionally, when some accurate classifiers suddenly become inaccurate, the system decides that the environment has changed. But this change might be not significant enough that the old optimal policy still applies to achieve the goal state. A more natural way is to carry out the old optimal action sequence after the environmental change and check if the goal state is still achieved. This requires a change of the explore/exploit framework such that an extra exploit trial is needed after the goal state is achieved in one exploit trial. Otherwise, the system needs to keep a record of every action sequence and store the optimal one, which is also possible but involves additional memory operations. As XCS has no internal memory, external memory is needed to store the goal state and the number of action steps towards it.

E. Reinforcement Component

The reinforcement (or credit assignment) component distributes the incoming reward among the classifiers. For sequential tasks, updates only occur in the previous time step's action set [A]$_{-1}$ because they make use of the prediction array on the following time step (see equation (2)), except that during the last trial of an episode, both [A] and [A]$_{-1}$ are updated. General *Widrow-Hoff learning rule* is used:

$$p_j(a) \leftarrow p_j(a) + \beta_p(P - p_j(a)) \tag{1}$$

where $p_j(a)$ is the system prediction of classifier j in [A]$_{-1}$ if its action a is performed, and $0 < \beta_p \leq 1$ is the learning rate controlling the prediction updates. P is the weighted sum of the previous time step's reward r_{t-1} and the maximal system prediction $P(a_i)$ for action a_i:

$$P = r_{t-1} + \gamma \max_i P(a_i) \tag{2}$$

where γ is the discount rate [8].

F. Rule Discovery Component

A rule discovery component applies a Genetic Algorithm (GA) to the classifiers to update the current knowledge. In sequential tasks, GA invocations occur in [A]$_{-1}$ where the Niche GA is triggered, which is a restricted mating scheme only happening among related classifiers [9]. This process is described as :

$$\frac{\Sigma_{x\in[A]}(t - GA_x) \times numerosity(x)}{\Sigma_{x\in[A]}\, numerosity(x)} > \theta_{GA} \tag{3}$$

where t is the current time step and GA_x is the time step when the classifier x was created but never been in such an $[A]$ or was last in an action set in which the GA was invoked. The $numerosity$ of x indicates the number of classifiers with the same conditions and actions. θ_{GA} is a threshold.

When Niche GA is triggered, two parent classifiers are selected with a probability proportional to their fitnesses, usually *Roulette Wheel Selection* is used in practice [7]. Standard crossover and mutation operators perform on them to get two offspring classifiers. Then, GA subsumption takes over to check if the condition part is logically subsumed by the condition of an accurate and sufficiently experienced parent [22].

The fitness calculation is the main difference between accuracy-based XCS and traditional strength-based LCS. When a classifier j is created, its initial prediction is denoted as p_j^0, prediction error as ϵ_j^0 and fitness as F_j^0. Then, p_j is updated by equation (1), ϵ_j updated by equation (4), and F_j updated by equation (5).

$$\epsilon_j \leftarrow \epsilon_j + \beta_e(|P - p_j| - \epsilon_j) \tag{4}$$

where ϵ_j is the prediction error of classifier j, β_e is the error learning rate, $|P - p_j|$ is the target prediction error towards which ϵ_j is updated.

$$F_j \leftarrow F_j + \beta_f(\kappa_j' - F_j) \tag{5}$$

where β_f is the fitness learning rate, classifier j's fitness F_j is updated towards its relative accuracy κ_j' calculated by equation (6) and (7):

$$\kappa_j = \begin{cases} 1 & \text{if} \quad \epsilon_j < \epsilon_0 \\ \alpha(\epsilon_j/\epsilon_0)^{-v} & \text{otherwise} \end{cases} \tag{6}$$

where ϵ_0 is a threshold that decides all classifiers' accuracies are equal if their prediction errors are below it, or decreased by α and v. Once $\kappa_j, j \in [A]$ have been updated, each classifier's relative accuracy κ_j' is updated by:

$$\kappa_j' = \frac{\kappa_j \times numerosity(j)}{\Sigma_{x\in[A]}\, \kappa_x \times numerosity(x)} \tag{7}$$

which normalizes the accuracies so that they sum up to 1. In short, a classifier's fitness is an inverse function of its prediction error, which is ignored if below ϵ_0.

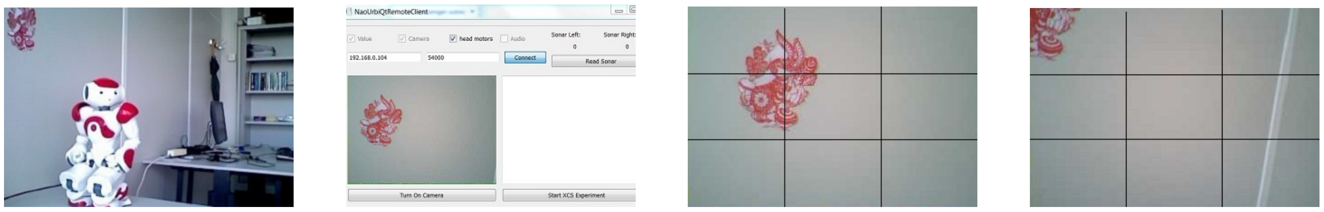
IV. Experiments

A sequential decision task is designed to test the performance of our modified XCS on the NAO robot in dynamic environments. The goal of the task is to make the NAO learn the combination of actions to find a target object and relearn the strategy as fast as possible when the environment is changed.

However, the experiment of XCS on real robots needs to consider more practical issues than computer simulations, e.g., the initialization of the robot's state which should be done reliably by the robot itself for several times, and the precision of its sensory readings, also its battery level that guarantees the experiment running for several minutes. Although NAO is a mobile robot, its localization and navigation are difficult issues by themselves in an open environment. Therefore, we decided to test XCS's performance in a task that does not require NAO's mobility. The following parts describe the experiment environment for the NAO and parameter settings for XCS, followed by experiment results and analysis.

A. Environment

The NAO is placed on a table, with power connected, see Fig.2(a). The forehead camera is used as the environmental input (320x240 resolution, 10 fps) of XCS and the action of the NAO is to move around its head which has two degrees of freedom. A GUI shows what the NAO sees in real time and has a control over the experiment, see Fig.2(b). The goal state of the NAO is defined as to find a red object and make part of it in the central view field (see Fig.2(c)). After several explore trials, the NAO is expected to learn a policy to quickly find the target object.



(a) NAO on a table (b) GUI (c) One goal state (d) Part of target found

Fig. 2. Experiment environment of the NAO searching task

1) State Representation: Every input image is segmented into 9 parts (see Fig.2(d)), each part represented by one binary bit which takes value 1 if there are more than M red pixels in that part and 0 otherwise. For example, the bit string representation of Fig.2(d) is 100000000. Similarly, Fig.2(c) is represented as 110110000.

2) Action Representation : The actions are encoded as 00 (move up), 01 (move right), 10 (move down), 11 (move left). The NAO head can pitch or yaw with a range of -0.67 to 0.51 radial ($-38.5°$ to $29.5°$) and -2.079 to 2.079 radial ($-119.5°$ to $119.5°$) separately. On each movement of the head, 0.1 radial is changed on current state with a speed of 0.05 radial/s. A boundary is defined as $[-pitch, pitch] \times [-yaw, yaw]$. Whenever the boundary is reached by the NAO, it fails the trial and is reset to the initial state to start a new trial.

3) Reward : A reward of 1000 is given when the fifth bit of the binary input string is 1, which represents the center part of the view field. In other cases, no reward is given to '0' bits and a reward of 100 is given to every other '1' bit. This should help the NAO to reach the goal state after only part of the target is seen.

4) A trial : An experiment consists of several trials and the NAO starts a trial from (0,0) position, with an empty initial [P] of classifiers or loads a population of classifiers from a file. It takes an image and sets its state using the aforementioned representation. While the goal state is not achieved or the boundary is not touched, the NAO keeps taking one action and capturing one image right after the action. Otherwise, it terminates this trial and returns to the initial state, starts a new trial or ends the experiment. In case of endless searching, the trial ends anyway after N_s steps.

B. Parameter Settings

The parameter settings take a reference of Wilson's *woods2* example [7] (see Table I). The main difference is the separate learning rates that β_p is for prediction update, β_ϵ for error update and β_f for fitness update. All of them are initialized with the default learning rate 0.2 and can be changed in dynamic experiments. Other changes of the parameters include the population size limit and GA threshold, both are decreased due to much less experiment trials compared with the thousands of trials in *woods2*. In our experiments, the movement boundary is $[-0.5, 0.5] \times [-0.5, 0.5]$, in other words, it is a 11×11 maze in which the NAO always starts from $(0, 0)$.

TABLE I
XCS PARAMETER SETTINGS FOR NAO SEARCHING TASK

Parameter	Notation	Value
Population size	N	100
discount factor	γ	0.7
Prediction learning rate	β_p	0.2
Error learning rate	β_ϵ	0.2
Fitness learning rate	β_f	0.2
Crossover probability	χ	0.8
Mutation probability	μ	0.04
Accuracy criterion	ϵ_0	10
Accuracy falloff rate	α	0.1
Accuracy exponent	v	5
Prediction initial	p^0	10
Error initial	ϵ^0	0
Fitness initial	f^0	0.01
Trial step limit	N_s	20
Hash probability	$P_\#$	0.6
GA threshold	θ_{GA}	5
Deletion threshold	θ_{Del}	20

In case of endless search, the maximal step in one trial is no more than $N_s = 20$.

C. Dynamic Environments

The environment can be changed when the NAO evaluates that it has learned the optimal policy to achieve the goal in one exploit trial, otherwise, it just keeps learning in explore trials. However, the robot does not change the environment by itself in this scenario, so the text-to-speech function of NAO is used to request the human operator to change the target object's position.

Equivalently, the orientation of the NAO's body is changed to simplify the operation. Several marks are made on the table to control the change. The complexity of this task can be changed easily with different spacial relations between the NAO and the target image.

D. Results

The first experiment proves that consistent actions within our explore/exploit framework outperforms all the others. Based on this result, a relearning task shows that a higher prediction learning rate plays a key role when the old knowledge does not apply any more. All the results are the average

TABLE II
AVERAGE TIME SPENT IN ONE TRIAL

Condition	Time (s)
uniform actions within Wilson's framework	34.2
semi-uniform actions within Wilson's framework	30.5
semi-uniform actions within our framework	24.4
consistent actions within Wilson's framework	20.2
consistent actions within our framework	19.5

TABLE III
POPULATION OF CLASSIFIERS

Condition	Action	Prediction	Error	Fitness	Exp.	Num.
1001#0000	01	10	0	0.01	0	1
#0###0##0	11	355.7	95.1	0.99	13	5
1#00#0###	00	453.6	116.2	0.94	11	1
0##00#00#	01	262.6	40.1	0.41	16	9
0#0#01##0	01	524.9	139.4	0.62	14	3

of 10 runs, each of which has 20 explore/exploit trials and 2 test trials.

1) Different actions within two frameworks : In this learning problem, 2 steps left are needed to reach the goal state. Different actions and explore/exploit frameworks are tested under the same parameters, Fig.3 gives the average steps per trial and Table II compares the average time spent in one trial. The worst case is the pure random action exploration which is quite inefficient for robotic tasks because learning happens even when the state of robot does not change. Semi-uniform actions improve the result with half chance of greedy actions. Furthermore, consistent actions make the learning faster that it steadily decreases from the 10th trial (see Fig.3(a)). Due to that the NAO keeps performing the same action until its state changes, it maintains much less macroclassifiers to solve the problem, see Fig.3(b). Also, an average around 5 steps happen a lot which probably means a failure trial where the boundary of the wrong direction is reached. Our framework enables a better performance that the NAO asks for a change and reacts to it immediately when the learned knowledge still applies to the new environment. In this case, turning one step left achieves the goal state.

2) Relearning with different learning rates : First, the NAO was trained with a 4 steps task, 2 steps up and 2 steps left. Table III gives some of the result classifiers. The first rule is generated by covering and not updated yet. The second rule is the most experienced one and has the highest fitness. It advocates action '11' (turning left) when no part of the red target is found. The third rule tells that when the left up corner has part of the target object, action '00' (turn up) is preferred.

Then, the changed environment needs 3 steps right to reach the goal state, which means the old knowledge can not solve the new problem and even misleads the actions in the initial state. In the beginning, the NAO always turned left using the second rule. After reaching the left boundary for several times, the second rule became inaccurate and the fourth rule in Table III took in charge, suggesting turn right. The last rule also encourages turning right and has a higher prediction because this state is closer to the goal state.

Despite of the rules, Fig.4 shows that the prediction learning rate of 0.8 performs better than the default 0.2 and stably converges around the 15th trial.

V. CONCLUSION

This paper presents a version of Extended Classifier System (XCS) on a humanoid NAO robot. The alternative framework between exploration and exploitation has been changed into a more efficient one, which is reasonable for robotic tasks. Despite of learning a complete, accurate, and maximally general map of an environment, the XCS utilizes external memory to store the current optimal policy which is supplementary to the traditional XCS without internal memory. In the exploit trials, not only greedy actions are selected, but also the goal state is remembered to evaluate the current environment. This guarantees that relearning happens only when necessary. Furthermore, the prediction learning rate is found to have more influence on the performance than the error learning rate and the fitness learning rate in dynamic environments.

Our research opens the door of further applications of LCS on humanoid robots. In future work, more advanced computer vision algorithms are needed to enrich the sensory input. Also, complex combinations of behaviors are required which can be obtained by increasing the environment's complexity. And in such an environment, the reliance on a priori knowledge should be unnecessary that the state boundaries should be decided by the system itself rather than be programmed by human operators.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] T. Mitchell, *Machine Learning*. McGraw-Hill,New York, 1997.

[3] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. A Bradford Book, Apr. 1992.

[4] P. L. Lanzi, "Learning Classifier Systems: then and now," *Evolutionary Intelligence*, vol. 1, pp. 63–82, 2008.

[5] J. H. Holland and J. S. Reitman, "Cognitive Systems based on Adaptive Algorithms," *SIGART Bull.*, no. 63, pp. 49–49, Jun. 1977.

[6] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, Mar. 1994.

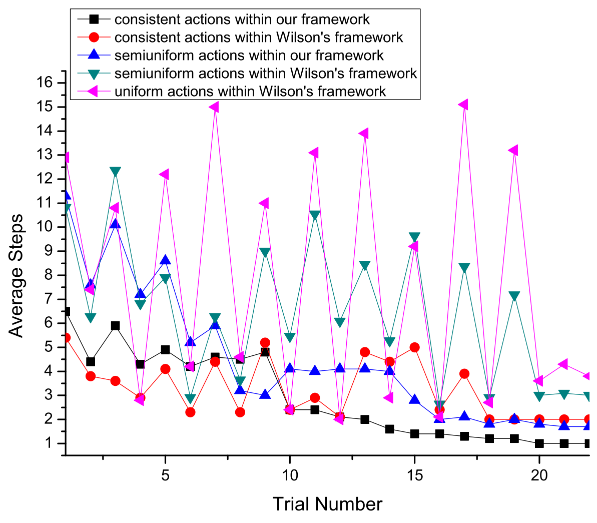
[7] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, Jun. 1995.

[8] T. Kovacs, *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. London, UK: Springer, 2004.

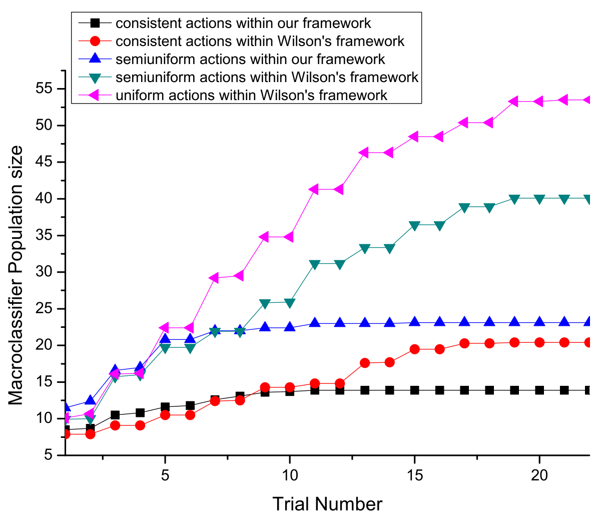
[9] S. W. Wilson, "Generalization in the XCS classifier system," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Madison, Wisconsin, USA: Morgan Kaufmann, pp. 665–674, 1998.

[10] D. Katagami and S. Yamada, "Interactive Classifier System for Real Robot Learning," in *Proceedings of 9th IEEE International Workshop Robot and Human Interactive Communication*, pp. 258 –263, 2000.

[11] J. Hurst, L. Bull, and C. Melhuish, "TCS Learning Classifier System Controller on a Real Robot," in *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, ser. PPSN VII. London, UK: Springer-Verlag, pp. 588–600, 2002.

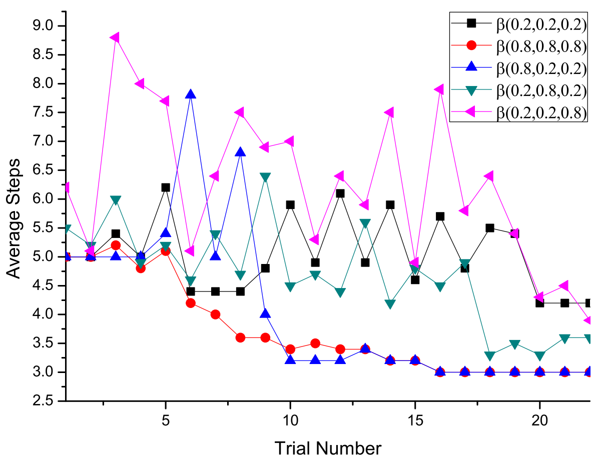


(a) Average steps in each trial

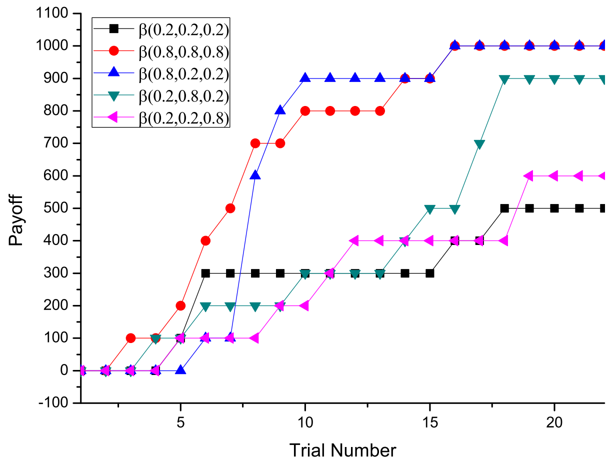


(b) Average population size in each trial

Fig. 3. Experiment results using different action selections and explore/exploit frames



(a) Average steps in each trial



(b) Average payoff in each trial

Fig. 4. Relearning with different learning rates $\beta(\beta_p, \beta_\epsilon, \beta_f)$

[12] M. Dorigo and M. Colombetti, *Robot Shaping*. Cambridge, MA, USA: MIT Press, 1998.

[13] Y. Iwakoshi, T. Furuhashi, and Y. Uchikawa, "A Fuzzy Classifier System for Evolutionary Learning of Robot Behaviors," *Applied Mathematics and Computation*, vol. 91, no. 1, pp. 73 – 81, 1998.

[14] L. Bull and J. Hurst, "A Neural Learning Classifier System with Self-Adaptive Constructivism," *The 2003 Congress on Evolutionary Computation*, vol. 2, pp. 991 – 997, Dec 2003.

[15] M. Studley and L. Bull, "X-TCS: Accuracy-Based Learning Classifier System Robotics," in *The 2005 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2099 – 2106, Sept 2005.

[16] P. Musilek, S. Li, and L. Wyard-Scott, "Enhanced Learning Classifier System for Robot Navigation," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3390 – 3395, Aug 2005.

[17] R. C. Moioli, P. A. Vargas, and F. J. Zuben, " Analysing Learning Classifier Systems in Reactive and Non-reactive Robotic Tasks," Berlin, Heidelberg: Springer-Verlag, pp. 286–305, 2008.

[18] L. Bull and J. Hurst, "ZCS Redux," *Evolutionary Computation*, vol. 10, no. 2, pp. 185–205, Jun. 2002.

[19] M. Troj and O. Unold, "Self-adaptation of Learning Rate in XCS Working in Noisy and Dynamic Environments," *Computers in Human Behavior*, vol. 27, no. 5, pp. 1535 – 1544, 2011.

[20] P. L. Lanzi and D. Loiacono, "XCSlib: The XCS Classifier System Library," 2009.

[21] P. L. Lanzi, S. W. Wilson, M. V. Butz, and T. Kovacs, "How XCS Evolves Accurate Classifiers," in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, pp. 927–934, 2001.

[22] M. V. Butz and S. W. Wilson, "An Algorithmic Description of XCS," in *Advances in Learning Classifier Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 267–274, 2001.

[23] C. Stone and L. Bull, "For real! XCS with continuous-valued inputs," *Evolutionary Computation*, vol. 11, no. 3, pp. 298–336, 2003.