4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp173-178)

Using a Coevolution Mechanism with a Dyna Architecture for Parameter Adaptation in XCS Classifier Systems

Chung-Yuan Huang[1,3], Chuen-Tsai Sun[3], Yen-Wei Chu[2,3]
[1)] Department of Computer Science and Information Engineering
[2)] Department of Information Management
Yuanpei Institute of Science and Technology
306 Yuan Pei Road
Hsinchu, Taiwan 30015, ROC
[3)] Department of Computer Science
National Chiao Tung University
1001 Ta Hsueh Road
Hsinchu, Taiwan 300, ROC

Abstract: - We propose a co-adaptive approach to control coevolution-based eXtended Classifier System (XCS) parameters. Taking advantage of the on-line incremental learning capabilities of XCS, solutions that completely address target problems can be produced. A coevolution model allows two XCS systems to operate in parallel to solve target and parameter setting problems simultaneously. Since our approach only requires small amounts of information on performance metrics during early run-time stages, it requires little time to become efficient in terms of latent learning. Test results indicate that our proposed system outperforms comparable models regardless of the target problem's stationary/non-stationary status.

Key-Words: - CA-XCS, Co-Adaptive, Coevolution, Dyna, Parameter Adaptation, Parameter Setting Problem

1 Introduction

Learning classifier systems (LCS) [2] were first introduced by John H. Holland (1975), who is also known as the father of genetic algorithms (GA). They use a combination of GA and reinforcement learning (RL) [8] to build adaptive rule-based systems that learn via on-line experiences [3, 11]. Depending on the architecture, LCS may be viewed as either an extended GA application or an RL algorithm. The GA component is responsible for comparing performance for the purpose of identifying better rules to replace unsuitable rules. The RL component is responsible for distributing credit among rules and resolving rule conflicts (e.g., distinguishing between appropriate and inappropriate rules).

Originally, LCS was not considered analyzable due to the complex nature of component interrelationships [3]. A renewed interest in LCS occurred after 1995, when Wilson proposed his eXtended Classifier System (XCS) [9] based on classifier prediction accuracy. A number of new models and applications have been presented since that time. Wilson retained most of Holland's original ideas and architecture, but also made several substantial changes that gave XCS at least four advantages [3, 9, 10]: a) easier analyzability; b) the ability to deal with complex problems (e.g., optimization issues) that had previously been considered unsolvable; c) the addition of a robust generalization mechanism capable of generating compact, complete, maximized, and accurate solutions [11]; and d) the capability to use various representations to specify classifiers [6, 7].

In the same manner as evolutionary computations (EC), the setting of parameters determines if XCS is capable of generating optimal or near-optimal solutions and its level of efficiency. All of the currently available approaches [1] to solving the parameter setting problem associated with LCSs have important drawbacks requiring improvement and modification. Our proposed co-adaptive approach, which is based on the coevolution concept and Dyna architecture [8], takes advantage of the incremental on-line learning capability of LCSs to produce solutions that completely cover a target problem.

The system simultaneously adapts parameters according to current learning performance and state. As shown in Figure 1, the framework consists of two LCSs. Main-XCS is responsible for solving the external target problem and meta-XCS is responsible for adapting internal parameters. We used the Dyna architecture to acquire the parameter control capability of meta-XCS in a short time period. Dyna uses an internal world model to save real experiences that are obtained during learning and uses them for an intensive latent learning process that shortens training time and speeds up the construction of a complete set of solutions. In [5], Lanzi showed that a combination of Dyna and XCS (Dyna-XCS) was capable of greatly enhancing learning performance. For the present study we used Dyna-XCS to a) solve the slow-learning problem of the adaptive parameter control approach (which requires a long training period) and b) significantly enhance parameter control stability.

To solve the parameter control problem common to LCSs, we established a framework in which main-XCS and meta-XCS operate in parallel. Solutions co-evolve as the systems cooperatively adjust parameters to a given target problem. There are two advantages to using coevolution to solve the parameter setting problem: many benefits of the self-adaptive parameter control approach are maintained without expanding the target problem's original search space, and the premature convergence problem that often accompanies this approach is avoided.

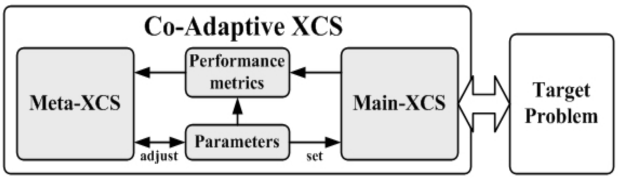


Figure 1. Co-adaptive XCS framework.

2 XCS Overview

In the same manner as traditional LCS, XCS is a problem-independent and adaptive machine learning model. As shown in Figure 2, XCS has four components: a finite classifier population, a performance component, a reinforcement component, and a rule discovery component. Stored classifiers control the system via a horizontal competition mechanism and perform tasks by means of vertical cooperation. The performance component governs interactions with the target problem. The input interface (detector) is used to transmit the current target problem state to the performance component and to determine dominant classifiers according to an exploration/exploitation criterion. Through the output interface (effector), any action advocated by dominant classifiers is executed and receives feedback. The reinforcement component (credit assignment component) uses an algorithm similar to Q-learning [8] to update the reinforcement parameters of classifiers that advocated the output action. Finally, the rule discovery component uses a GA to search for better or more general classifiers and to discard incorrect or more specific classifiers.

When running XCS, performance metrics are used to observe system performance and to express classifier populations. Kovacs [11] divided these performance metrics into two categories: performance measures and population state measures. Three well-known on-line metrics for measuring performance in research environments and real-world XCS applications are performance ρ, system error, and population size. Performance ρ and system error are used to measure XCS learning capability according to results from target problem interactions. As one would expect, the population size performance metric (defined as the number of macro-classifiers in a classifier population) belongs to the category of population state measures. It is used to measure XCS learning quality.

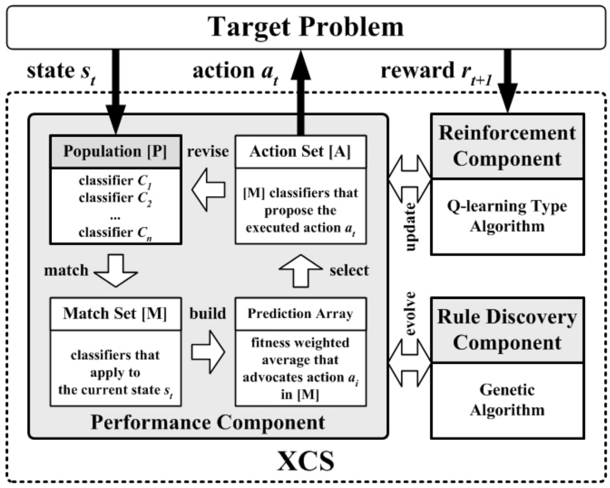


Figure 2. XCS architecture.

3 Co-Adaptive XCS (CA-XCS)

Our decision to use one XCS to adjust the parameters of other XCS system was based on its ability to deal with complex problems, especially its ability to represent various parameter control strategies [3, 6, 7]. We used XCS to capture relationships and changing parameter patterns between parameter control strategies and to observe their effects on target problem. Its principal features include a) the combined advantages of adaptive and self-adaptive parameter control approaches that allow for the use of a coevolution model to simultaneously solve a target problem and parameter setting problem, and b) reduced time requirements for becoming efficient via a latent learning process that uses small pieces of information about performance metrics in the early stages of a run.

3.1 The Model

A schematic of the co-adaptive XCS (CA-XCS) architecture is shown in Figure 3. Its four principal components are main-XCS, meta-XCS, performance metrics, and parameters. Similar to the basic XCS, the main-XCS component is responsible for interacting with and solving the target problem. The meta-XCS integrates Dyna architecture with XCSμ [4], which is especially useful in stochastic environments where the results of actions are affected by uncertainty. The meta-XCS component learns parameter control and adjustment strategies very quickly. The performance metrics component is responsible for collecting, recording, and evaluating the main-XCS component. The parameter component stores all parameters that need adjustment by the meta-XCS component and assigns updated parameters to the main-XCS component.

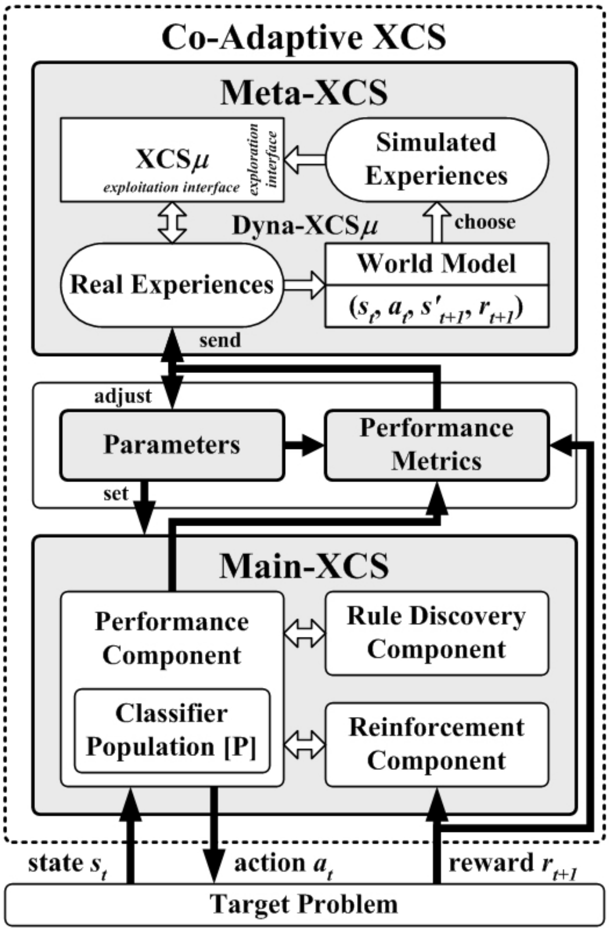


Figure 3. CA-XCS architecture detail.

3.2 Meta-XCS Component

After running the main-XCS component for n trials (e.g., n = 50 in 6-MP), assuming a current discrete time step of t, the meta-XCS component receives an input message (s_t) ← (Π_t, ρ, ε_{sys}, σ_{pop}) transmitted by the parameter and performance metrics components. In addition to the current parameter settings Π_t affecting the main-XCS component, message s_t also contains measures of the main-XCS component's performance ρ, ε_{sys}, and population states σ_{pop}. Based on the information in s_t, the XCSμ in the meta-XCS component determines an appropriate parameter control action a_t and instructs the parameter component to update the corresponding parameter p_i in parameter setting Π_t. Next, the meta-XCS component receives a s_{t+1} message and r_{t+1} feedback in the form of a reward or penalty from the performance-metrics component. The meta-XCS component uses r_{t+1} for reinforcing learning and for storing the (s_t, a_t, s_{t+1}, r_{t+1}) information within the Dyna architecture. During intervals between parameter control actions, the meta-XCS component uses these records for latent learning. The cycle continues until the target problem is solved by the main-XCS component or a user-requested stop criterion is met.

3.3 Meta-XCS Dyna

As shown in Figure 4, Dyna uses the XCSμ exploration interface to perform latent learning, and the parameter control operation is executed through the XCSμ exploitation interface. Theoretically, the latent learning and practical parameter control operations of meta-XCS can be processed simultaneously, but in practice, a higher priority is assigned to the parameter control operation in the meta-XCS component in order to decrease potential conflicts and to meet the hardware restrictions of sequential processing. Therefore, latent learning is delayed until parameter control operations are fully executed. However, we believe that the arrangement takes advantage of system idling time to improve parameter control and learning performance.

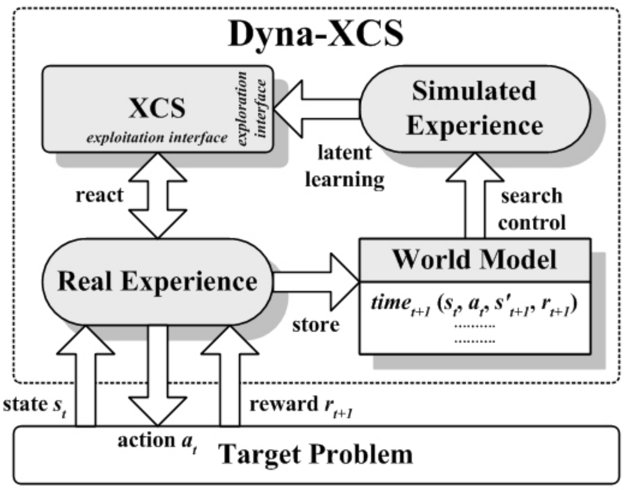


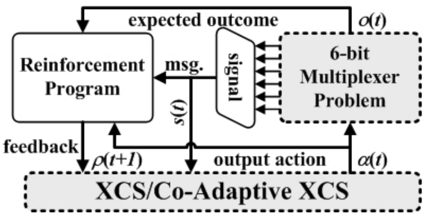
Figure 4. Dyna-XCS architecture.

4 Experiments

In LCSs, the mutation operator plays an important role in learning performance and target problem solution quality. During the early stages of a run, the mutation operator provides novelty by moving classifiers within the search space. A faster mutation rate helps speed up the rule discovery component. As an essential background operator during the later stages, mutation ensures some probability of finding a better solution. A slower mutation rate helps fine-tune existing classifiers without disturbing runs or decreasing learning performance. However, it is difficult to predetermine the optimal mutation rate for a given target problem or to dynamically adjust the mutation rate during every stage of a run.

4.1 6-bit Multiplexer Problems (6-MP)

Given the restrictions just described, we experimented with a co-adaptive parameter control approach in the form of the 6-bit multiplexer problem (6-MP)—a version of the well-known benchmark single-step problem for machine learning in general and LCSs in particular [9]. As shown in Figure 5, the input message signal transmitted to LCSs consists of a string of six binary digits in which the first (version A) or last (version B) two bits (called address bits) represent a binary index and the remaining bits represent data bits. The expected outcome is the value of the indexed data bit. For example, the expected outcome of “111110” in version A is 0, since the first two bits (11) represent index 3—the fourth bit following the address. The expected outcome of “010001” in version B is 1, since the second bit preceding the address is indexed.



$$o_a(t) = f_{6\text{-}mp}(b_0,b_1,b_2,b_3,b_4,b_5) = \overline{b_0}\,\overline{b_1}b_2 + \overline{b_0}b_1b_3 + b_0\overline{b_1}b_4 + b_0b_1b_5 \quad \textit{(version A)}$$

$$o_b(t) = f_{6\text{-}mp}(b_0,b_1,b_2,b_3,b_4,b_5) = \overline{b_4}\,\overline{b_5}b_0 + \overline{b_4}b_5b_1 + b_4\overline{b_5}b_2 + b_4b_5b_3 \quad \textit{(version B)}$$

Figure 5. The 6-bit Multiplexer Problem (6-MP)

6-MP is considered challenging because of its non-linear characteristic, yet it yields many useful generalizations that help in comparing learning performance in various models. During each cycle, the 6-MP produces signals by randomly setting all six bits. Expected outcomes are computed as single bits from the generated signals, which are transmitted as input messages to the LCS on request and returned as output actions that are compared with expected outcomes. A positive feedback score of 1,000 means that a reward was returned to the LCS for reinforcement; a feedback score of 0 means that a penalty was returned. During the run, the 6-MP continues to produce 6-bit messages with similar probabilities as the LCS tries to learn the correct mapping relations between signals and expected outcomes—thus developing an optimal solution.

With the exception of the mutation rate, the default parameter for our experiments was $N = 800$, $\beta = 0.2$, $\alpha = 0.1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, $P_{\#} = 0.33$, $p_I = 10$, $\varepsilon_I = 0$, $F_I = 0.01$, $p_{explore} = 0.5$, *doGAsubsumption* = true, *doActionSetSubsumption* = true. All results discussed in this report represent an average of 30 runs. XCS performance metrics were recorded for each trial and computed as average moving window numbers in the last 50 trials.

4.2 Stationary Problem Experiment

In the first experiment we used the 6-MP to examine whether the meta-XCS component of CA-XCS could adjust the main-XCS component mutation rate. We used three original XCS (one each with fixed mutation rates of 0.01, 0.05, and 0.09) in the stationary problems to determine whether CA-XCS performance and learning quality was the best among all XCS version.

Performance metrics data from CA-XCS and three comparative XCS at fixed mutation rate of 0.01, 0.05, and 0.09 in the stationary 6-MP experiment are shown in Figure 6. When the XCS mutation rate = 0.09, the population metric was usually the worst and the performance ρ metric the best among the three XCS versions; furthermore, the system error metric decreased very quickly. When the XCS mutation rate = 0.01, performance ρ was the worst metric, the system error metric decreased very slowly, and the population size metric was the smallest among the three XCS versions; furthermore, the number of classifiers was nearly half that of the XCS at the fixed mutation rate of 0.09.

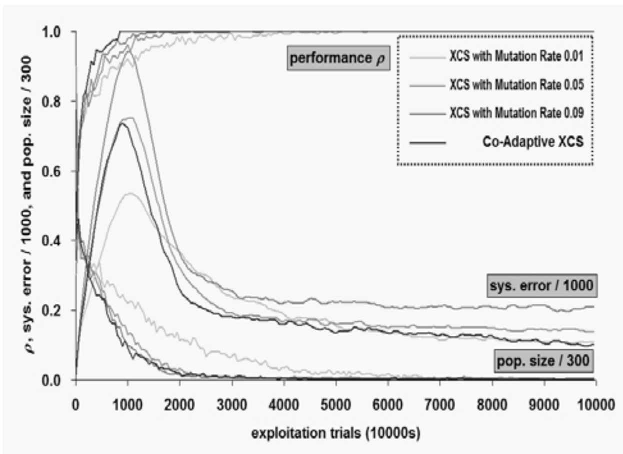


Figure 6. Data for CA-XCS performance ρ, system error, and population size performance metrics for the stationary 6-MP problem (version A) and other XCS versions with

fixed mutation rates of 0.01, 0.05, and 0.09.

When the CA-XCS was applied to 6-MP learning, the performance and learning quality dilemmas were easily avoided. As shown in Figure 6, it took between 850 and 900 trials for CA-XCS to completely learn correct mapping relationships between the 6-MP input messages and expected outcomes. For the population size metric, the CA-XCS matched or outperformed the XCS at a fixed mutation rate of 0.01—that is, after 10,000 trials, approximately 21 classifiers were available for forming an optimal solution.

Results from average mutation rate adaptation processes for CA-XCS in the stationary 6-MP (version A) are shown in Figure 7. The results support those from previous studies on EC mutation rates. During early run trials, the meta-XCS component tended to increase the mutation rate in the main-XCS component in order to produce classifiers capable of learning correct mapping relationships between 6-MP input messages and expected outcomes at maximum speed. In the later trials, it tended to decrease the mutation rate in order to fine-tune existing classifiers and to obtain an optimal solution.

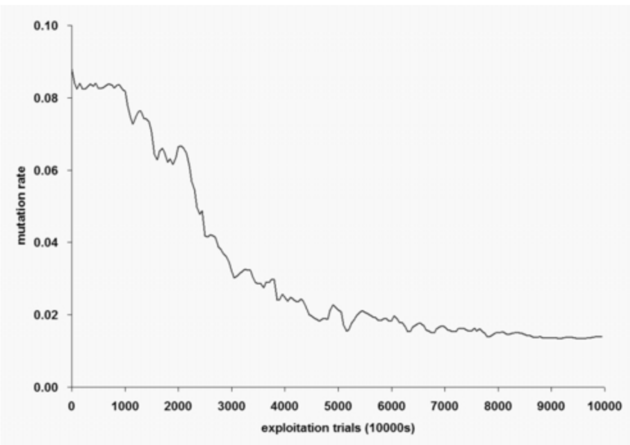


Figure 7. Variation in CA-XCS mutation rate adaptation process for the stationary 6-MP problem (version A).

As shown in Figure 7, a constant high-to-low oscillation was observed in the mutation rate of the main-XCS component. It appears as though the meta-XCS component of CA-XCS tested the influence of a lower mutation rate on the main-XCS component during early run trials and tested the influence of a higher mutation rate on the main-XCS component during later trials. Mutation rates were abandoned if they exerted negative impacts on performance and learning quality. In these cases, the meta-XCS component returned to the opposite end of the mutation rate range and rapidly mastered a link among the mutation rate, performance ρ metric, population size metric, and similar situations via a latent learning mechanism.

4.3 Non-Stationary Problem Experiment

Our second experiment was similar to the first except that the second made use of 6-MP versions A and B (Figs. 8 and 9). For this experiment we ran 20,000 trials—10,000 that were similar to the first experiment and 10,000 in which the input signal bit sequence was abruptly changed from version A to B. In the version B run, the two address lines were moved from the initial (b_0b_1) to final input signal bit (b_4b_5) position. Whenever the bit sequence underwent a sudden change during the second 10,000 trials, the CA-XCS had to re-generalize the existing classifiers and rebuild an appropriate solution. The goal was to determine whether or not the CA-XCS could quickly reestablish a proper mutation rate following an abrupt change in the problem environment, recover the original learning performance and population size state, and still rebuild an optimal solution.

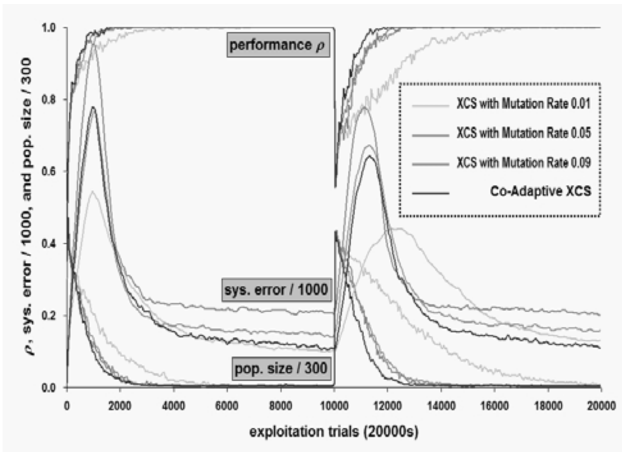


Figure 8. Performance ρ, system error, and population size performance metrics of CA-XCS and other XCS versions with fixed mutation rates of 0.01, 0.05, and 0.09 for the non-stationary 6-MP problem (versions A and B).

Details of CA-XCS performance are presented in Figure 8. Regardless of the performance metric, the system outperformed XCS at the fixed mutation rates of 0.01, 0.05, or 0.09. At a fixed mutation rate of 0.09, the performance ρ and system error metrics for the CA-XCS outperformed those from the XCS. At a fixed mutation rate of 0.01, the population size metric for the CA-XCS was similar to that from the XCS. An optimal solution aimed at the 6-MP version B was rebuilt after 20,000 trials.

Figure 9 has two mutation rate peaks, the first before 2,500 trials and the second between 10,000 and 12,400 trials. Each peak reflects the time required by the CA-XCS to learn from the beginning. According to these experimental results, the CA_XCS is capable of handling non-stationary problems at a high performance level.

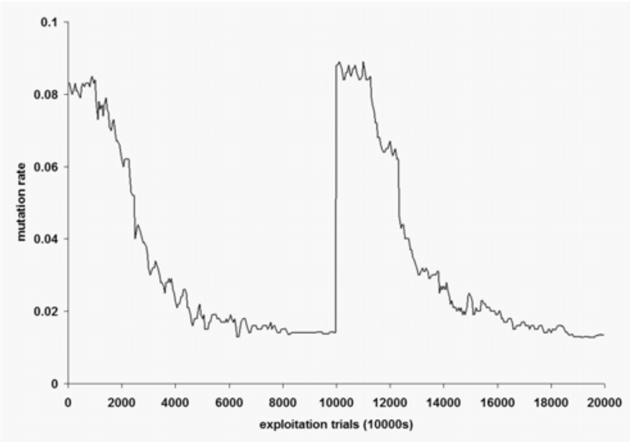


Figure 9. Mutation rate adaptation for the CA-XCS in the non-stationary 6-MP (versions A and B).

5 Conclusions

Previous studies of ECs and LCSs describe the search for robust or optimal parameter sets for target problem solutions as a time-intensive trial-and-error task requiring large amounts of computation resources. Different parameter values are essential for inducing an optimal balance between exploration and exploitation at different run stages. In response to the common problem of setting parameters for practical applications, we extended the original LCS with a parameter control approach in order to enhance performance and learning stability.

Our proposal for a co-adaptive approach to LCS parameters is based on a coevolution process and a Dyna architecture. The approach takes advantage of the on-line learning capabilities of LCSs; solutions produced in this manner cover entire target problems. Results from our experiments show that the co-adaptive approach was successful in terms of setting parameters according to target problem properties. In both stationary and non-stationary problem experiments, the system outperformed the models it was tested against.

Acknowledgement:

This work was supported in part by National Science Council, Taiwan, Republic of China under grant NSC 92-2524-S-009-004 and NSC 93-2520-S-009-003.

References:

[1] Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2 (1999) 124-141.

[2] Holland, J.H.: *Adaptation in Natural and Artificial Systems. MIT Press* (1992); University of Michigan Press, Ann Arbor (1975)

[3] Holmes, J.H., Lanzi, P.L., Stolzmann, W., Wilson, S.W.: Learning Classifier Systems: New Models, Successful Applications. *Information Processing Letters*, Vol. 82 (2002) 23-30.

[4] Lanzi, P.L., Colombetti, M.: An Extension to the XCS Classifier System for Stochastic Environments. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO'99*). Morgan Kaufmann (1999) 353-360.

[5] Lanzi, P.L.: An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, Vol. 7, No. 2 (1999) 125-149.

[6] Lanzi, P.L.: Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO'99*). Morgan Kaufmann (1999) 337-344.

[7] Lanzi, P.L.: Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO'99*). Morgan Kaufmann (1999) 345-352.

[8] Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press (1998).

[9] Wilson, S.W.: Classifier Fitness Based on Accuracy. *Evolutionary Computation*, Vol. 3, No. 2 (1995) 149-175.

[10] Wilson, S.W.: Generalization in the XCS Classifier System. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R.L. (eds.): *Genetic Programming 1998: Proceedings of the Third Annual Conference* San Francisco. Morgan Kaufmann (1998) 665-674

[11] Kovacs, T.: What Should a Classifier System Learn and How Should We Measure It? *Journal of Soft Computing*, Vol. 6, Nos. 3-4 (2002) 171-182.