1998 International Computer Symposium
Workshop on Artificial Intelligence
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

COOPERATIVE-BASED LEARNING CLASSIFIER SYSTEM: A MAN-MACHINE MODEL APPROACH

Wei-Hua Andrew Wang[1], Hsiao-Lan Fang[2] and Pei-Fang Tsai[3]

[1]Institute of Industrial Engineering, Tunghai University, Taiwan, R.O.C.
Email: wangwh@ie.thu.edu.tw

[2]Artificial Intelligence Application Group, Electrical & Control Department, China Steel Corporation, Taiwan, R.O.C.
Email: hsiaolan@ksts.seed.net.tw

[3]System Engineering Division, Manufacturing Control System Department, Mechanical Industry Research Laboratories, Industrial Technology Research Institute, Taiwan, R.O.C., Email: 860256@mirl.itri.org.tw

ABSTRACT

In this paper, we propose a *Cooperative-Based Learning Classifier System* (CBLCS) which combines the genetic-based learning classifier system with a *Man-Machine Model* (MMM) to come up with better performance than traditional classifier system. The proposed MMM consists of two constituents: *Conflict Resolution Mechanism* and *User Interface Component*. In particular, we use a *Reliability Probability* to decide whether to take the user's advice or not when conflicts occur. An elevator-scheduling problem is then used to demonstrate the performance of the proposed CBLCS approach. Results show that the CBLCS can improve not only the learning speed but also the solution quality.

1. INTRODUCTION

A system cannot survive long in a dynamic environment unless the system keeps pace with environment. However, the cost of development a system is usually not trivial. As a result, how to extend the life of a system is becoming important. If a system can adapt itself to a new environment, then the system is supposed to live longer. A number of machine-learning systems [1] have been proposed during the past two decades. These systems can more or less adapt themselves to the environment through their embedded learning mechanism. One of these branches is Classifier System (CS) [2, 3, 4], which is a kind of new genetic-based machine learning mechanism in the evolutionary computation field. Because of its learning ability, so the word "learning" was usually preceded to the name, to make: "Learning Classifier System" (LCS). The initiator of research in Genetic Algorithms (GAs) and Classifier System (CS) is John Holland. Holland published the book "**Adaptation in Natural and Artificial Systems**" [5], and from then on many papers and dissertations began to be published by different researchers. For example, [6] proposed a Zeroth Level Classifier System (ZCS), which preserved much of Holland's original framework but simplified it for implementation by introducing the matching set and the action set. Later, [7] continued to present XCS, in which each classifier maintained a prediction of expected payoff, but the classifier's fitness was given by a measure of the prediction's accuracy. Recently, [8] proposed a cooperative learning classifier system, which elaborated the key similarities between learning classifier systems and artificial neural networks (ANN). He suggested that an ANN, with genetically evolved receptive fields, is a type of learning classifier system.

Although these studies have provided valuable frameworks in order for a system to be adaptable, none mentions the cooperative architecture of human and LCS. This paper describes a *Cooperative-Based Learning Classifier System* (CBLCS) [9] that combines the advantages of both the learning ability of XCS and the valuable experience of the user. We testes the architecture on an elevator-scheduling problem and find that the performance is promising. The results of this study may be of interest to researches who want to enhance the real world application of the classifier systems.

The paper is organized as follows. Section 2 presents the Learning Classifier Systems (LCS). Section 3 describes the architecture of CBLCS. In section 4, the MMM of CBLCS is proposed. Experiments on the elevator-scheduling problems are tested in section 6. Finally, conclusions are given in section 7.

2. INTRODUCTION TO LEARNING CLASSIFIER SYSTEMS

In this section, we will briefly introduce the concept of LCS, including Holland's original prototype of CS and Wilson's ZCS and XCS. The last two frameworks improve the first one by simplifying to increase understandability and performance. Moreover, it is easier for us to implement.

2.1 What Are Classifier Systems?

The first descriptions of classifier systems appeared in [10]. This led to *Cognitive System One* (CS-1) [11], the first application of a classifier system. A simple classifier system maintains a set of rules, called *classifiers*, encoded by the combinations of '0', '1' and '#' (don't care). Figure 1 [3] shows a simple framework of the simple classifier system, which is composed of three components, namely, *Classifier System* (performance level), *Bucket Brigade* (learning level) [12], and *Genetic Algorithm* (discovery level) [5, 13]. The classifier system is the main component that uses a set of *If-condition-Then-action* classifiers to control the whole system by receiving message from the environment, matching messages with the classifiers, and sending messages to the environment for an corresponding action accordingly. Meanwhile, the Bucket Brigade mechanism awards those matched classifiers by distributing the feedback from the receivers, such as the environment. Moreover, the Genetic Algorithm injects new classifiers into the system by the genetic reproduction, crossover and mutation operators. What is the essence of a CS? In a word, it is a rule-based, incremental learning system with rules evaluated through experience and evolved by a Darwinian process.

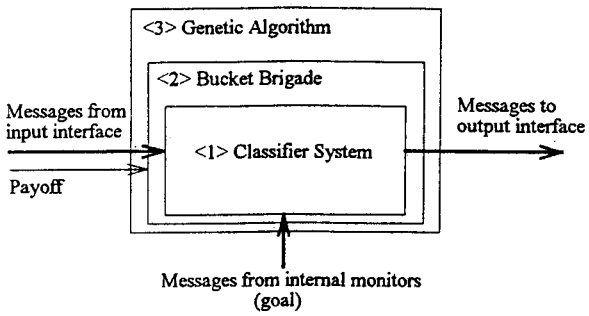

Figure 1. Architecture of a Simple Classifier System

2.2 What Is ZCS?

Though Holland's architecture is nearly perfect, its framework is very complicated. In other words, it is too perfect to implement. As a result, simplification is important. [6] simplifies Holland's CS and proposes the *Zeroth-level Classifier System* (ZCS). Here, "Zeroth-level" refers to the reduction of the CS to its minimum number of operational components. The zeroth-level preserves the essence of the CS without the complex extensions existing in CS research. The architecture of ZCS is shown in Figure 2, where [P] denotes the population set, [M] denotes the matching set, and [A] and $[A]_{-1}$ denote the current and previous action set, respectively. ZCS uses [M] to group classifiers of the same action and calculates the sum of strength of the same group. Action selection is conducted by *Roulette wheel selection*, the most popular selection method, and classifiers of the winning action are put in [A]. So, even classifier with weakest strength has the chance to be selected because of the effect of the action-based selection. In addition, ZCS introduced a new GA operator, *Covering*, which produces a new classifier by matching the input message with some proportions of *don't care* ('#') in the condition part, and selecting a random action in the action, when there is no classifier fires, i.e., [M] is empty. For more details of ZCS, and other difference between CS and ZCS, please refer to [6].

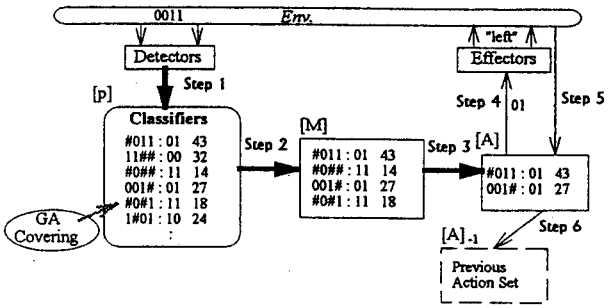

Figure 2. Architecture of ZCS

2.3 What Is XCS?

[7] further proposes the XCS to investigate the issues of rule strength and credit-assignment. In XCS, each classifier maintains a prediction of expected payoff, but the classifier's fitness is given by a measure of the prediction's accuracy. XCS tries to extend the original strength-based fitness to a more exact accuracy-based fitness credit assignment. [7] introduces three new parameters in his XCS: *prediction* (***p***), *prediction error* (ε) and *fitness* (*f*). ***p*** is used in classifier's action-selection; *f*, GA's parent-selection; ε, fitness updating. In ZCS, general, but inaccurate, classifier survives; however, in XCS, because of the effect of the more accurate fitness-prediction mechanism, selective pressure against too general classifiers, but toward classifiers that are both accurate and maximally general [14]. In other words, it can distinguish with the difference between the accurate classifiers and the over-general ones. Essentially, the architecture of XCS is similar to that of ZCS; however, the more sophisticated selection that accuracy-based makes possible. Also, GA works in the whole population set in ZCS; whereas, it works only in the match set in XCS. Moreover, the learning algorithm used in ZCS is Bucket Brigade, but XCS further combines it with *Q-learning* [15]. An example adopted from [7] to illustrate the action selection procedures of XCS is shown in Figure 3., in which a fitness-weighted average of the prediction of classifiers advocating action a_i, $P(a_i)$,

is calculated for each a_i. Then the $P(a_i)$ values are placed in a prediction array, and an action of the maximum $P(a_i)$ is selected. For more details of updating the p, ε and f, and other differences between ZCS and XCS, please refer to [7, 16, 14].

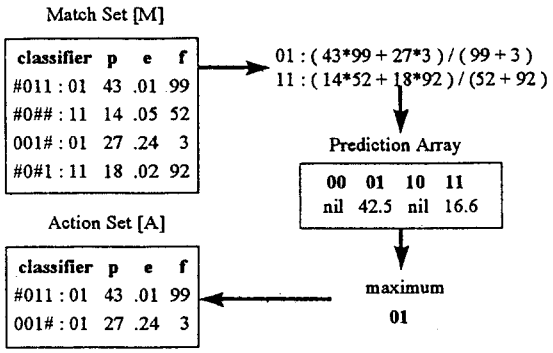

Figure 3. Accuracy-based Selection of XCS

3. ARCHITECTURE OF CBLCS

CBLCS is based on the cooperative architecture of XCS (agent) and MMM (user). XCS is a suitable agent for three primary reasons: the learning ability, the creativity and the rule-based representation. First, the learning ability in XCS is its reinforcement mechanism using rewards from environment, which can lead the system toward the maximum reward. Second, the creativity means the implement of genetic algorithm, which can enable the system to jump from some absorbing states. Third, the rule-based representation can be translated as the human knowledge, just as that of the expert systems do. As to the MMM, we design two constituents, the *Conflict Resolution Mechanism* and the *User Interface Component*, to let the user join the learning process. The user interface component has two functions, to interpret the agent's decision and to encode the user's knowledge. When the agent accepts the situation from environment, the user interface component will show the current situation, display all the decisions from the agent and the prediction values about those decisions. On the other hand, the conflict resolution mechanism resolves the conflicts accordingly between the agent and the user.

The scenario is going as the description underneath. When the environment has changed and the agent is required to make a decision, the Detectors of agent receive the status from environment at first. Then, the Detectors encode the status into a binary string and pass it to the Classifier-Set. When the Classifier-Set gets the binary string, it would compare all the condition part of its classifiers with the binary string and collect the copy of matched ones to a Match-Set. Next, the Genetic Algorithm may invoke. After those processes, all the possible decisions the agent could make are in the Match-Set, and these data will be passed to the User-Interface, which can decide the way to display those data, do some prediction calculation and wait for the user's response. If the User agree the prediction, then the agent will make the decision by the original Match-Set. Otherwise, the user's decision will flow into the Conflict Resolution Mechanism with the original Match-Set. If the user wins, then the user's decision will be encoded as a new classifier to form a new Match-Set to substitute the old one. If not, then the original Match-Set will not be changed. The Conflict Resolution Mechanism will accumulate both the reliability of the user and that of the agent. In later section, the detail design about the Conflict Resolution Mechanism will be discussed. For now, the overall architecture of CBLCS is shown in Figure 4, and the corresponding processing steps are listed after it.

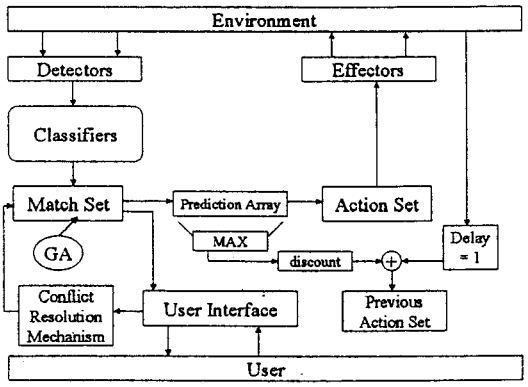

Figure 4. Architecture of CBLCS

1. The *Detectors* check whether environment sends input message.
2. Each condition part in the Population Set [P] is compared with the detector string. If the bit at every position in a classifier's condition matches the corresponding bit in the detector string, put the classifier into the current Match Set [M].
3. Predict [M]'s possible action and display it to the user interface.
4. If the user does not agree with the system's action input a new action via the *User Interface Component*. Otherwise, go to step 6.
5. Within the *Conflict Resolution Mechanism*, make a choice based on the *Reliability Probability*. If the user wins the competition, replace [M] with the user's rules.
6. Update each $P(a_i)$ in the *prediction array*, by calculating the fitness-weighted average of the prediction of classifiers advocating a_i, where a_i denotes the i^{th} action. If no rule supports a specific action i, then just put a "nil" in $P(a_i)$.
7. From the *prediction array*, select the largest $P(a_i)$.
8. The corresponding a_i is sent to *Effectors*, and carried

out in the environment.

9. Multiplying a_i by a discount γ, we have B. Store B in a *Bucket* and subtract portion of p, ε and f from the corresponding parameters of [A].
10. If the environment produces reward, then distribute the reward to the previous Action Set $[A]_{-1}$.
11. Adding the reward from environment to the *Bucket*, we have P.
12. Modify p, ε and f of $[A]_{-1}$ in terms of P.

4. MAN-MACHINE MODEL

We propose the MMM to build a cooperative and competitive mechanism by which the user can work with the system together. The rationale of the approach is that the whole system is able to learn to make certain judicious choice through MMM, which is composed of two parts as below.

4.1 Conflict Resolution Mechanism

By *conflict resolution* we refer to the choice made when the decisions between the system and user are inconsistent. To judge which one, the system or the user, is dependable is not an easy task. When the system's performance is poor, we would like to count more on the user. On the other hand, we tend to count less on the user to preserve the original decision when the system performs well. In order to resolve the conflicts between the system and the user, we use a *Reliability Probability* (**RP**) to decide whether to accept the user's decision or not. In other words, we use the RP to measure the relative importance between the system and the user. Upon receiving a reward from the environment, it is the right time to update the RP. If the reward is a negative value, the RP will decrease; nevertheless, the RP will increases.

4.2 User Interface Component

The communication between the system and the user is conducted with the help of *user interface component*. We need to consider the knowledge representation problem here in the design of the user interface, i.e., how to represent the user's knowledge. For reducing the incompatibility between the system's classifiers and the user's input, we decide to use the same representation as that of the system. This can avoid misunderstanding induced by unnecessary knowledge transformation from the user to the system. The user needs only to key in his suggestions directly in a format similar to that of the classifier, combinations of 0, 1, and #. However, other people may use an user-friendlier interface, e.g., natural language front end or graphic user interface, at an expense of some forms of transformation.

5. EXPERIMENTS

The proposed CBLCS architecture was tested on a set of elevator-scheduling problems to verify whether the MMM can help to (1) increase the convergence speed, and (2) decrease the number of human interruption during the learning process.

5.1 The Elevator-scheduling Problem

The elevator scheduling is a high-dimensional, dynamic problem. There is not a single algorithm that can optimize the schedule because the requirement changes dynamically. Figure 5 shows a status of the elevator problem at a specific time step, where the black rectangle denotes the elevator, the plus symbol denotes an entrance requirement, and the minus symbol denotes an exit requirement on a specific floor. The problem is to schedule the four elevators in a building with 10 floors. The elevators are to carry customers from one floor to another under the following assumptions:

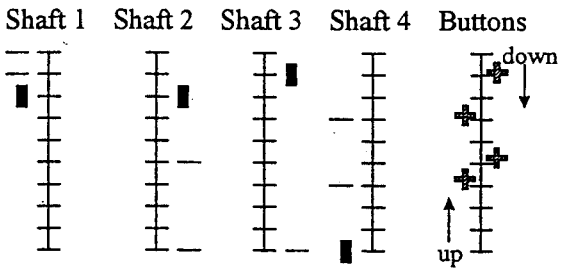

Figure 5. The Elevator-scheduling Problem

1. There is no inter-relationship between any two elevators. I.e., the status of one elevator will not affect that of the other, and vice versa.
2. Each entrance requirement contains no more than 6 customers and no less than 1 customer.
3. There are at most 6 requirements under service for a single elevator.

According to assumptions 2 and 3, we know that the maximum capacity of an elevator is 36. Moreover, the elevator-related parameters used in the experiment are assumed below:

1. Number of floor: 10 (from 0 to 9),
2. Number of elevator: 4 (from 0 to 3),
3. Average time of a requirement: 5 seconds,
4. Speed of elevator: 1 floor / second,
5. Time taken by each entrance or exit: 0.5 second /

person.

Other requirements not yet mentioned are all assumed to be a uniform distribution.

5.2 Representation

We used 12 bits to encode an input message. Bit 0, 4 and 8 encode information about the first elevator; bit 1, 5 and 9, the second, etc. Each input message is further divided into three groups; the detail is presented as below.

1. *The first group, i.e., the first 4 bits, refers to the distance between each elevator and its corresponding requirement.* If the distance is less than or equal to 2 floors, then assign the corresponding bit to 1; otherwise, assign it to 0.
2. *The second group represents the direction between each elevator and its corresponding requirement.* If both are of the same direction, then assign the corresponding bit to 1; otherwise, assign it to 0.
3. *The last four bits refer to the customer's longest waiting time of each elevator.* If the waiting time is less than or equal to 15 units of time, then assign the corresponding bit to 1; otherwise, assign it to 0.

We further used 2 bits to encode the output message: 00 denotes elevator 0; 01, elevator 1, etc. Parameters used in CS and GA are mostly adopted from [7]. For example, parameters related to the initial accuracy are: p = 10, ε = 0, and f = 10. The learning rate (γ) equals to 0.71, discount factor (β) 0.2, and value of the fraction of classifier deletion (δ) 0.1. The GA time threshold(θ) is set to 25, crossover rate(χ) 0.5, mutation rate(μ) 0.33, and the population size 50. A detailed description of these parameters is given in [7].

6. RESULTS AND DISCUSSIONS

The experiments were set up into two groups: the standard mode and the contrast mode. The latter consisted of one experiment conducted purely by the user and the other purely by the XCS. As to the former, we conducted nine experiments to test on a variety of man-machine learning processes. Each of the 11 experiments lasted for 30 units of time, and, in turn, each unit of time was composed of 10 requirements. That is, we interrupted and collected data from the learning process per ten job requirements (randomly generate), and repeated it for 30 times each test. When we interrupted the process, we got another copy of the classifier set. Altogether we have 30 copies of classifier sets for each test. The reason to cut the learning process into 30 pieces is trying to find the 'trajectory' of the learning characteristics. Then we further produced 5 set of fix data as the input; you might call it a fix scenario. Next we took away the learning mechanism and put those copies of classifiers collected from different phases into the fix scenario to get the performance purely by the XCS. In our experiment, the performance at each unit of time of the tests was measured by the customer's average waiting time and all the tests were conducted by the same person in order to have a stable environment.

The following nine figures presented the performance of CBLCS vs. XCS. Hereafter, we often refer CBLCS as MMM for the latter does not exist alone without being embedded in the former. From Figures 6-1, 6-2, 6-4, 6-5 and 6-7, we obtain that the overall performance of MMM is more stable than that of the XCS. Though the performance of MMM is rather unstable in the first half of Figure 6-8, it is approaching stable in the second half. The performance of MMM of Figures 6-3, 6-6 and 6-9 remains unstable even in the latter part of the time step. However, it is still more stable than that of the XCS. In general it is clear that, with the help of the MMM in the learning process, the CBLCS outperform the XCS. In order to observe more on the learning ability, we plotted the average and the standard deviation of the user's performance in Figure 7 in which we know that the user's common sense is far better than that of the XCS.

Then, we calculated the difference of the last 5 waiting times of the user and the corresponding waiting time of the XCS. Similarly, the difference between the user and those of the nine MMMs were recorded. Next, we average each of the five differences and put them in the column entitled convergence value as shown in Table 1. We can see more clearly that all of the convergence values of the MMM are very close to 0. In other words, their final performance is very similar to that of the user. Also, none of the MMM model's convergence value is larger than that of the XCS. So, we can say that the MMM does improve the performance of the XCS.

System	Convergence value
XCS	4.292
1stMMM	1.964
2ndMMM	0.76
3rdMMM	2.308
4thMMM	1.276
5thMMM	0.264
6thMMM	1.916
7thMMM	0.924
8thMMM	0.924
9thMMM	3.564

Table 1. Value of Convergence

Finally, we further plot the *RP* of the CBLCS with MMM in Figures 8-1 to 8-3. The initial value of the *RP* is

assumed to be 0.6. The system is told to increase the *RP* by 0.02 if the user makes good decision; otherwise, decreased it by 0.03. Here, we multiply the *RP* by 100 and plot 3 tests in a Figure. We found that the *RP* was generally descending in all the nine tests. In a sense, it is an indication that the system counts less on the user as time advances. In other words, the system does learn something from the user during the whole process and becomes more and more intelligent.

7. CONCLUSIONS

The proposed CBLCS architecture uses the idea of the cooperation between man and machine learning by combining the MMM into XCS. User contributes his knowledge through the user interface component, and the system then decides whether to take the advice from the user or not. The more the user participates in the system, the more he is satisfied with it. Thus, the whole system can adapt itself sooner than without the user's participation. CBLCS is then tested on the elevator-scheduling problem. The performance is promising in several respects.

1. CBLCS outperforms XCS as far as the stability is concerned.
2. The solution quality of CBLCS is better than that of XCS.
3. CBLCS learns experience from the user and accordingly avoid unnecessary trial-and-error during learning.

To sum up, with the addition of the MMM, CBLCS changes its role from the original decision-maker to a dual one, *decision-support* and *knowledge-acquisition*.

1. Decision Support System: If the user is a newcomer or unfamiliar with the environment, the system plays a role in decision support. For example, in a scheduling environment, in the absence of user's experience, the system learns the scheduling knowledge by reinforcement from the environment. The user can then use the schedule that still guarantees the quality to some extent. Otherwise, the novice may produce a terrible schedule due to his inexperience. As a result, in this sense we can say the CBLCS plays a decision support role.
2. Knowledge-Acquisition System: If the user is an expert, the knowledge acquisition role comes to play. For example, in a similar scheduling described above, an experienced user is supposed to contribute useful suggests during the scheduling process. The system can then make use of these suggestions as well as its embedded mechanism to learn how to schedule. As a result, we can say that part of the scheduling knowledge learned in the system is the accumulation of human experience. In this sense, the CBLCS can be regard as a knowledge acquisition tool.

8. REFERENCES

[1] Tom Mitchell, "Machine Learning", McGraw Hill, 1997.
[2] Holland, J. H. and K. J. Holyoak and R. E. Nisbett and P. R. Thagard, "Induction: processes of inference, learning, and discovery", The MIT Press, 1986.
[3] Holland, J. H. and L. B. Booker and D. E. Goldberg, "Classifier systems and genetic algorithms", the University of Michigan Cognitive Science and Machine Intelligence Laboratory Technical Report No. 8, April, 1987.
[4] Stewart W. Wilson and David E. Goldberg, "A critical review of classifier systems", Proceeding of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1989.
[5] Holland, J.H., "Adaptation in neural and artificial systems", The University of Michigan Press, 1975.
[6] Stewart W. Wilson, "ZCS: A zeroth level classifier system", Evolutionary Computation, Vol. 3, No. 2, 1994.
[7] Stewart W. Wilson, "Classifier fitness based on accuracy", Evolutionary Computation, Vol. 2, No. 1, 1995.
[8] Henry Brown Cribbs, III, "Cooperative learning classifier systems: a neural network approach", MS Thesis, Department of Engineering and Mechanics, Alabama University, 1995.
[9] Pei-Fang Tsai, "A study of classifier-based collaborate scheduling (CBCS) system", MS Thesis, Institute of Industrial Engineering, Tunghai University, 1997.
[10] Holland, J.H., Adaptation, In Rosen, R. and Snell, F. M. (Eds.), "Progress in Theoretical Biology IV", pp. 263-293, New York: Academic Press, 1976.
[11] Holland, J. H. and J. S. Reitman, "Cognitive systems based on adaptive algorithms", In D. A. Waterman & F. Hayes-Roth (Eds.), Pattern directed inference systems, pp. 313-329, New York: Academic Press, 1978.
[12] Goldberg, D. E., "Genetic algorithms in search, optimization, and machine learning", Addison-Wesley Publishing Company, Inc., 1989.
[13] Melanie Mitchell, "An introduction to genetic algorithms", The MIT Press, 1996.
[14] Stewart W. Wilson, "Generalization in the XCS Classifier System", in Genetic Programming: Proceedings of the Third Annual Conference, Madison, Wisconsin University, USA, 1998.
[15] Watkins, J. C. H., "Learning with delayed rewards", Unpublished doctoral dissertation, King's College, London, 1989.
[16] Stewart W. Wilson, "Generalization in XCS", Evolutionary Computing and Machine Learning, ICML, 1996.

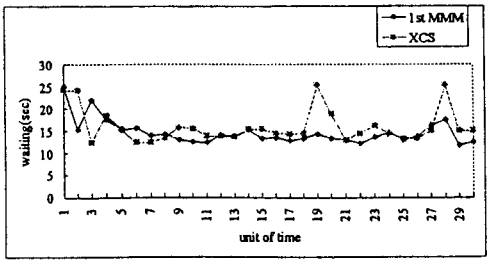

Figure 6-1. 1st MMM vs. XCS

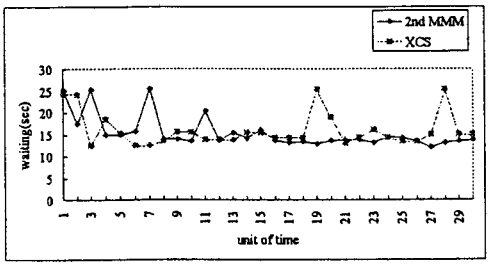

Figure 6-2. 2nd MMM vs. XCS

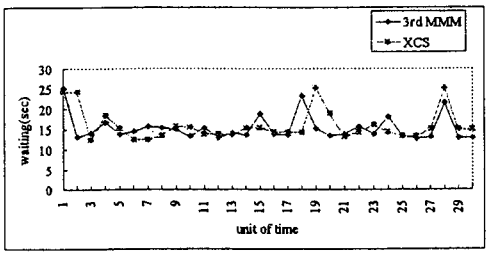

Figure 6-3. 3rd MMM vs. XCS

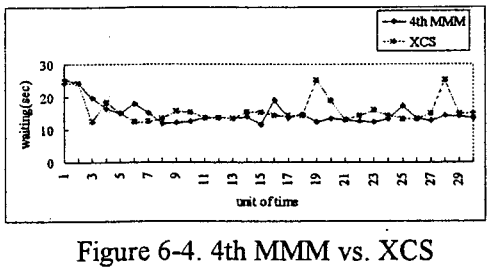

Figure 6-4. 4th MMM vs. XCS

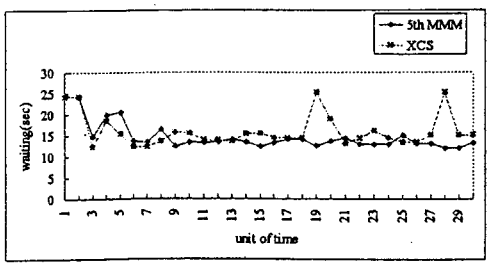

Figure 6-5. 5th MMM vs. XCS

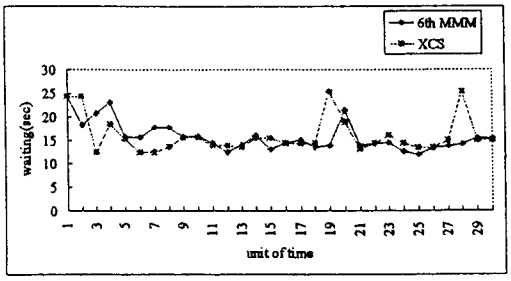

Figure 6-6. 6th MMM vs. XCS

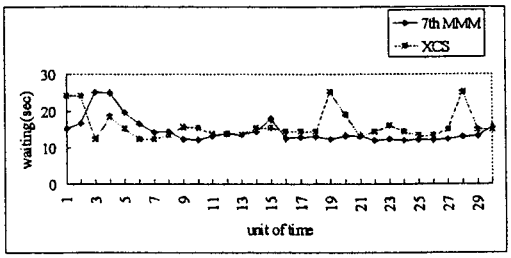

Figure 6-7. 7th MMM vs. XCS

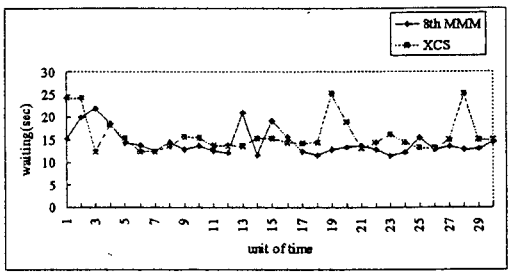

Figure 6-8. 8th MMM vs. XCS

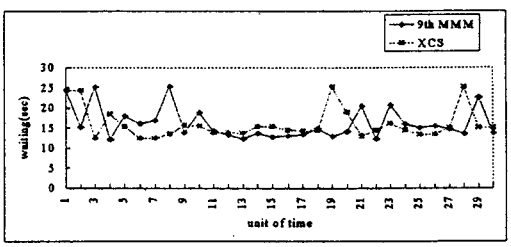

Figure 6-9. 9th MMM vs. XCS

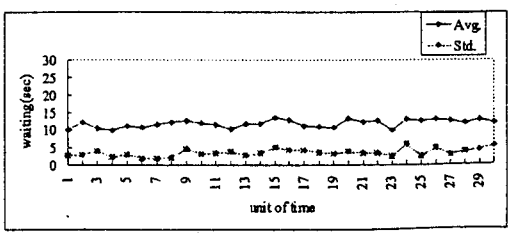

Figure 7. Average vs. Standard Deviation of Human User

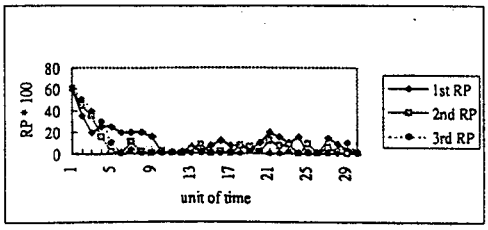

Figure 8-1 1st RP, 2nd RP and 3rd RP

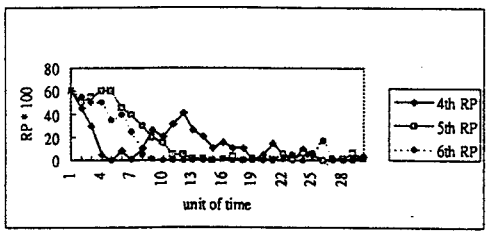

Figure 8-2 4th RP, 5th RP and 6th RP

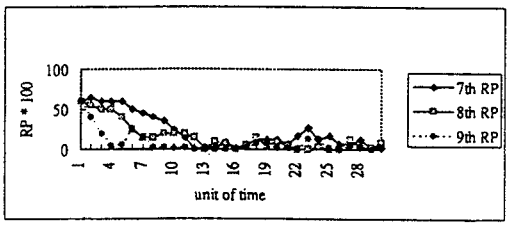

Figure 8-3 7th RP, 8th RP and 9th RP