博 士 論 文

Classifier System の解析と複雑な問題への応用に関する研究

Analysis of Classifier System and Its Application to Complex Problems

劉 淑梅
(Shumei Liu)

2006年9月

横浜国立大学大学院
環境情報学府

Doctoral Dissertation

Analysis of Classifier System and Its Application to Complex Problems

Classifier System の解析と複雑な問題への応用に関する研究

Shumei Liu

Thesis advisor: Professor Tomoharu Nagao

September 2006

Graduate School of Environment and Information Sciences,
Yokohama National University

Abstract

Recently, the research on Artificial Intelligence has been studied in various fields such as the autonomous agent control, Job Shop Schedule, data mining, forecasting for time series, and so on. The research has been focused mainly on complex problem that could not be simply solved by mathematical algorithms. The common objective is to get an optimal solution in its search space. However, in an enormous search space, it is hard to get a global optimization, or to get an optimization quickly. In this thesis, we focus on how to obtain an optimal solution efficiently for two different complex problems, respectively, and our effort is put mainly on Classifier System (CS). The solution is originated from Genetic Algorithm (GA) and Reinforcement Learning (RL), and suitable especially for complex environment.

First, we discuss the autonomous agent control. We consider the benchmark problem of Grid-Maze with Non-Markov property, and propose an adaptive XCSM system(AXCSM) on it. As related work, XCS (eXtended Classifier System) is an effective approach for Markov environment, but not for Non-Markov ones. By introducing a bit-register memory to XCS to record agent's experience, XCSM(XCS with Memory) has been verified to be suitable for Non-Markov environment. The classifier in XCSM is expanded with constant length of memory to that of XCS. Combining the current perception with its past experience, the agent can get optimal action. However, with the memory becoming longer, the search space will expand rapidly, and the performance of XCSM decreases quickly. Therefore, AXCSM is proposed by involving a hierarchical structure to adapt to the variable length of memory. In the first hierarchy, we learn a suitable memory length for a given perception by general XCS. The optimal action is then achieved in the second hierarchy by XCSM. The experimental results show that the adaptive XCSM converges the same to optimal policy as XCSM, but within shortened search space.

Next, we focus on another complex problem, the stock market forecasting. While analyzing a real stock market, we concentrate in general on qualitative changes first, and then on tracing

the quantitative changes. Therefore, it is necessary to give an indication of the changing direction, upwards or downwards, and then get the changed value as precise as possible. In order to forecast the real stock market, we propose a Hierarchical XCS (HXCS) model, which is composed of multiple local models. Each local model represents an individual agent. In the lower levels of the hierarchy, agents are trained by the XCS model to learn and forecast. These agents are only appropriate for some of the changing patterns in the time series data, and they fail to describe other changing patterns. For the upper levels of the hierarchy, RL is used to determine how to shift among those local models for a changing trend. With the hierarchical learning structure, multiple agents work alternatively and the limitation of a single agent can be overcome. To evaluate the prediction performance, we mainly adopt the prediction accuracy of changing tendency of the next time (up or down), which is measured by changing direction hit-rate. Besides the tendency prediction, we introduce a variable reward strategy on different prediction accuracy, to provide a value prediction. Experiments have been performed on several well-known stock indexes and stock markets. The results show that our proposal can achieve a higher performance when predicting the changing trend in the next day, and give a value predication.

In conclusion, two approaches are presented in this thesis. Each of them corresponds to a different kind of problems. The experiments show that each of them works well in dealing with the corresponding problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

To be truly helpful to humans, robots must be intelligent and able to function autonomously. To be considered intelligent, a robot should control in a complex and noisy environment without human aid. However, most of the robots lack both autonomy and intelligence, and they are either directly controlled by humans or preprogrammed. Therefore, the autonomous agent control has become an active area of research. Based on Genetic Programming (GP), the research on game playing [1], object movement [2], and obstacle avoidance navigation [3] has been pursued on Khepera, a mobile robot. For a simulation environment, [4, 5, 6] has been proposed based on Neural Network, GP and Reinforcement Learning respectively. The learning is pursued online by recognizing the environment dynamically, and acquiring the latest environment information. Then the objective centers on recognizing environment with dynamic method, and the bottle-neck lies on how to describe the environment with a knowledge-base as compacted as possible. For a simple problem, it is easy to realize; however, with the problem complexity increasing, the knowledge base expands rapidly. This will result in the performance of algorithm decreasing obviously. Despite the enormous approaches have been proposed, an ideal method on the autonomous agent control is also expected.

On another aspect, as computers, sensors and information distribution channels proliferate, there is an increasing amount of data created. However, the data is not much valuable if we just gather the signals without recognizing or analyzing them in advance. This is why the classification and prediction step in, extracting useful information from raw data. In this thesis, we mainly focus on prediction of time series. Examples of time series include

financial data series (stocks, indices, rates, etc.), physically observed data series (volcano eruption, weather, etc.), and mathematical data series (Fibonacci sequence, integrals of differential equations, etc.). The phrase time series refers to any data series, whether the data is dependent on a certain time increment or not.

Time series forecasting has several important applications. One is to prevent undesirable events or lessen the danger by forecasting it, as in [7]. It uses a method of SVM (Support Vector Machine) to forecast the flow of Mississippi River. The other is to benefit those people primarily in the financial markets from time series forecasting. Now many approaches and products are available for financial forecasting. In [8, 9], they use approach of SOM (Self-Organizing Map), NN (Neural Network), kNN (k-Nearest Neighbor) to predict the next varying trend to make maximum profit.

In this thesis, we focus on financial data forecasting as well. Furthermore, the same approach is applicable to other time series, directly or indirectly by changing the parameter setting.

In the two application fields mentioned above, the central technique is to construct a solution population to adapt to the problem. Thus, it is important to know how to identify and exploit a problem domain . As more difficult problems are addressed, the size of this population grows quickly and how to take advantage of the available knowledge becomes important. LCS (Learning Classifier System) is a machine learning technique which combines reinforcement learning, evolutionary computing and other heuristics to produce adaptive systems. Recently, LCS has been widely expanded from binary value [10, 11, 12] to real value environment [13], and has been applied in various fields such as data mining [14], stock market analysis [15, 16], and so on. The advantage of Classifier System has been approved greatly, especially for the complex problem. As for the Classifier System, we will introduce its principle and working strategy in Chapter 2.

In summary, we will discuss how to get the regularities on expressing a given problem, and define a suitable rule expression for the search space to get higher search efficiency in this thesis. Apparently, any approach provides a lot of flexibility when applied to a new problem. While accompanied with this flexibility, it is sometimes difficult to encode all the parameters perfectly, but just suits for one or a little special kinds of environment.

1.2 Research Procedure

In this thesis, first of all, we center on autonomous agent control. For simplification, we consider the benchmark problem of Grid-Maze with Non-Markov property, and propose an adaptive XCSM system (AXCSM) on it. As related work, XCS is an effective approach for Markov Environment, but not for Non-Markov ones. By introducing a bit-register memory to XCS to record agent's experience, XCSM (XCS with Memory) has been verified to be suitable for Non-Markov environment. The classifier in XCSM is expanded with constant length of memory to that of XCS. Combining the current perception with its past experience, the agent can get optimal action. However, with the memory becoming longer, the search space will expand urgently, and the performance of XCSM decreases as well. Therefore, AXCSM is proposed by involving a hierarchical structure to adapt to the variable length of memory. In the first hierarchy, we learn a suitable memory length for a given perception with general XCS. The optimal action is then achieved in the second hierarchy by XCSM. The experimental results show that the adaptive XCSM converges the same to optimal policy as XCSM, but within shortened search space.

Next, we focus on another complex problem, the stock market forecasting problem. When analyzing a real stock market, we concentrate in general on qualitative changes first, and then on tracing the quantitative changes. Therefore, it is necessary to give an indication of the changing direction, upward or downward, and then get the changed value as precise as possible.

In order to forecast the real stock market, we propose a Hierarchical XCS (HXCS) model, which is composed of multiple local models. Each local model represents an individual agent. In the lower levels of the hierarchy, agents are trained by the XCS model to learn and forecast. These agents are only appropriate for some of the changing patterns in the time series data, and they fail to describe other changing patterns. With the hierarchical learning structure, multiple agents work alternatively and the limitation of a single agent can be overcome. To evaluate the prediction performance, we mainly adopt the prediction accuracy of changing tendency of the next time (up or down), which is measured by changing direction hit-rate. Besides the tendency prediction, we introduce a variable reward strategy on different prediction accuracy, to provide a value prediction. Experiments have been performed on several well-known stock indexes and stock markets. The results show that our proposal can

achieve a higher performance when predicting the changing trend in the next day, and give a value predication.

1.3 Organization of the Thesis

This thesis is organized as follows: in Chapter 2, we reviewed the related work, mainly focused on Classifier System. In this chapter, the principle of Classifier System and its improvement version LCS, XCS, has been illustrated in detail. Its application and prospect in future is also outlined. Based on this chapter, we concluded that Classifier System is effective on learning for complex problem, and its prospect is valuable as well. In Chapter 3, as a benchmark of autonomous agent control, Grid-Maze problem has been involved, especially for Non-Markov problem. For this kind of problem, we introduced our proposal of adaptive XCSM for recognizing the Non-Markov problem with higher performance. In Chapter 4, we focused on a typical prediction problem in real world, stock market forecasting problem. A Hierarchical XCS has been proposed, and it accuracy is also verified. The last chapter, Chapter 5 is the conclusion of this thesis.

Chapter 2

Related Work

2.1 LCS; Learning Classifier System

Since the proposal of GA from 1970's, its improvement and application have been well known. As one instance, applying GA to the domain of Machine Learning, a predominant work called Classifier System is spread by involving production rules.

According to the two related but distinct approaches in GA, the LCS is also segmented into the Michigan approach [17] and Pittsburgh approach [18]. In the former, LCS contains a single population with limited size of classifiers, and each classifier is a production rule. In the later, it maintains a number of distinct populations, and each population is a single genotype. The Pittsburgh approach is more accordant with general GA; however, it is often computationally expensive and will be problematic when confronted with large populations. Thus, in this thesis, we involve the Michigan approach.

For a LCS, it composes of Production System, Learning Algorithm and Induction Algorithm generally. In this section, we first introduce the Production System of traditional LCS [17, 19, 20] by simulating its working process step by step; then the Induction Algorithm and Learning Algorithm will be explained respectively.

2.1.1 Production System

The LCS works as the following schedule in iteration.

Step1: Receive an input from the environment and translate it into a message.

Generally, the production rule is divided into two parts: a condition and an action, which

represents a simple stimulus-response system. While the classifier in LCS consists of one or two conditions, the first one represents perception of the environment, and the second one represents values from other classifiers. This enables one classifier to be triggered by other classifiers and thereby produces a sequence of classifier from a single input. If two conditions are provided by the classifier, the classifier can be created with "internal message". The action may use an additional tag bit to identify whether the action is to be performed on the environment or is to be transformed into the "internal message". Using internal messages, it is possible to establish a sequence of actions for complex determination. The condition and action are strings of characters from the ternary alphabet {0,1,#}. The '#' acts as a wildcard allowing generalization such that the classifier condition '1#1' matches both the input '111' and the input '101'. The symbol also allows feature pass-through in the action such that, in responding to the input '101', the rule "IF 1#1 THEN 0#0" would produce the action '000'.

Step2: Match the message against each classifier.

For any classifier, if its condition matches the current input, or any others on the message list, it becomes a member of the MatchSet. If no classifier is matched, create a new classifier to match the input and insert it into population, through the Induction Learning. The detail will be introduced at Chap. 2.1.3.

Step3: From the MatchSet, an suitable action is identified through a bidding mechanism and is pursued to environment.

Step4: If environment feedbacks, the value is assigned as reward, and allocated to those classifiers that provide the action. Otherwise, classifiers that provide the action in the current iteration devote payment as payoff, and pay it to those ones in previous iteration. As for the update detail, refer the Learning Algorithm in Sect. 2.1.2.

Step5: Periodic delete classifiers with poor performance, and create new classifier with GA, which obeys the operation of Induction Algorithm.

2.1.2 Learning Algorithm

The environment that a LCS learns within can be divided into two classes. The first one is composed of those environments where an environmental feedback is returned on each stimulus-response step. These will be termed single-step environments. The second one contains those environments where the agent is only given a reward after a number of stimulus-

response iteration, where it successes. These will be termed multi-step environments. The learning procedures for these two kinds of environments are not the exactly same, but with a slight difference. We will explain them respectively now.

For both single-step and multi-step environment, when a reward is received from environment, it will be assigned to the classifiers with the current optimal action, based on a bidding mechanism [21]. Whilst, in multi-step environment, before receiving a reward, at each iteration, all classifiers that have been selected to propose an output message will devote a proportion of their strength as bidding as Eq.(2–1). Then the bidding is paid back to the classifiers that have activated them in the previous iteration. All the classifiers at the previous iteration and at the present iteration have been identified. This enables the strength passed back as payoff to those classifiers activated indirectly, avoiding parasite classifiers from gaining undeserved reward. We call this "Bucket Brigade" algorithm, each classifier is much like the middleman in a commercial chain.

$$bidding = \beta * strength * specificity \tag{2–1}$$

In addition to the reward and payoff values, two kinds of "Tax" are imposed on the population. The "Life Tax" is to remove a fixed proportion of the current strength of all classifiers as Eq.(2–2). The "Life Tax" is introduced so that classifiers which do not match within any environmental niche have their strength decreased until they come under deletion pressure.

$$strength = strength * (1 - lifeTax) \tag{2–2}$$

A "Bid Tax" is applied to classifiers in MatchSet and is to remove a fixed proportion of strength as Eq.(2–3). The "Bid Tax" increased the rate of strength reduction of classifiers that propose actions, but produce poor payoff or reward.

$$strength = strength * (1 - BidTax) \tag{2–3}$$

2.1.3 Induction Algorithm

The Learning Algorithm is used to identify the usefulness of one classifier. But it could not replace poor classifiers with better classifiers or improve upon their usefulness. To realize this, we use the mechanism of Induction Algorithm. Firstly, the GA was conceived as the primary rule induction algorithm, by creating a new offspring and deleting a poor one. Beside GA, an

induction operator has been proposed. When no classifier matches to the current situation, a new classifier will be created with the appropriate condition and an often randomly created action.

Apart from this, if the current classifiers have a history of poor performance, a new classifier from the current classifiers is created and inserted into the population to replace the poor one.

2.2 XCS

2.2.1 ZCS

Based on the traditional LCS, a more simplified LCS [22] and then ZCS [23] (Zeroth level Classifier System) have been proposed by Wilson to increase the understandability and performance. ZCS is composed of a simple single condition, a single discrete action, a single strength measure, and reliance upon an implicit bucket-brigade. It uses ActionSet rather than individual rules. All members of MatchSet that have the same action as the selected rules form the ActionSet. Then Learning Algorithm is treated on the ActionSet.

The result of ZCS has been shown that it is capable of deriving optimal performance, at least in a number of well-known test problems. However, it appears to be particularly sensitive to some of its parameters as well. Frankly, ZCS has been somewhat superseded by XCS.

Deriving form Wilson's Animate [22] and ZCS [23], XCS [24] has been proposed. Here, we overview the structure and operation of XCS briefly. The detail implementation has been provided at [25].

2.2.2 Production System

The classifier in XCS is defined as:

$$classifierXCS ::$$

$$c : Condition$$

$$a : Action$$

$$s : Strength$$

$$v : Value$$

$$n : Numerousity$$

$$Condition = (< 0 > | < 1 > | < \# >)^*$$

$$Action = (< 0 > | < 1 >)^*$$

$$Strength ::$$

$$p : Prediction$$

$$\varepsilon : Error$$

$$\kappa : Accuracy$$

$$f : Fitness$$

$$Value ::$$

$$e : ActionSetEstimate$$

$$x : Experience$$

$$g : GAIteration$$

$$Numerosity = Integer$$

In general LCS, the 'Strength' identifies payoff prediction of the classifier. Instead of that, in XCS a series of parameters have been involved to measure it. First, the measure 'Prediction' has been introduced to describe the reward that the classifier system should expect to receive, if the condition proposed by the classifier is matched and its action is selected. A second measure 'Accuracy'(κ) is a calculated likelihood of the classifier obtaining the reward identified by 'Prediction'. It is derived from a third maintained value 'Prediction Error'(ε), which is a proportion representing the absolute difference between the prediction of the classifier and the reward or payoff received over a period of time. Finally, the measure 'Fitness' is derived from 'Accuracy' and is the accuracy of the classifier relative to that of other

classifiers that propose the same action in which the classifier fires. Generally, Prediction is used for action selection, and Fitness is used for selection in GA, with other values being maintained to effect the calculation of Fitness. The most signification difference between XCS and most other classifier systems is that rule fitness for the GA is not based on the payoff prediction but on the accuracy of predictions in payoff.

In XCS, an additional parameter 'Numerosity' is introduced as a convenience factor. Because of the GA and covering operator in XCS, many duplicated classifiers will be created. Because the classifiers in XCS is reward sharing method(all the classifiers in ActionSet will receive same reward or payoff value), it is possible to remove these duplicate classifiers and simply maintain a count of the number of duplicates. Then they represented by a single classifier instance within the population, with the parameter 'Numerosity'. On the other hand, besides the duplicated classifiers, subsumed classifiers also exist frequently in XCS. If condition part of one classifier can be subsumed by another classifier's, and they own the same actions, it is also benefit to remove the more specific classifier, but sum the 'Numerosity' to the general one. Clearly, the overall population size must be calculated in terms of the sum of the numerosity of all the classifiers, rather than the number of classifier instances. To simplify, we define these duplicated classifiers with 'Numerosity' as 'MacroClassifier'. Henceforth all the references to 'Classifier' will refer to 'MacroClassifier' if a distinction is not needed. Otherwise, the 'MacroClassifier' and 'MicroClassifier' will be used.

With this structure, a single trial of the XCS is started as an exploration or an exploitation trial. The exploration/exploitation probability is generally set as 0.5.

Once the mode of operation has been decided, a new message is obtained from the environment. Then it is compared to all of the classifiers in the current population, and all the matched classifiers are added to create MatchSet. If no matched classifier exists, a new classifier is created with the covering operator, shown in Sect. 2.2.4.

Based on these classifiers in MatchSet, for each possible action that may be performed, PredictionArray is calculated as Eq.(2–4), as the fitness weighted average of the predictions of classifiers. This means that the classifiers with higher fitness will contribute more to predict that action.

$$predictionArray[a] = \frac{\sum_{cl \in M} p * f}{\sum_{cl \in M} f} \tag{2–4}$$

And then, XCS chooses an action to perform. The selection method depends upon the exploration/exploitation strategy adopted. If exploration mode is adopted, XCS selects an

action at random; otherwise, the action with the highest PredictionArray is selected. Then the selected action is pursued to environment.

For those classifiers in MatchSet, if its action is the same as the assigned action, it is identified to form ActionSet.

When a reward is received from environment, the classifier in ActionSet will update its parameters based on Learning Algorithm, illustrated in Sect. 2.2.3. Finally pursue Induction Algorithm on ActionSet, described in Sect. 2.2.4.

2.2.3 Learning Algorithm

Described as above, classifiers within the ActionSet are credited in one-step or multi-step environment. In these two kinds of environment, the classifiers in ActionSet of the current iteration are credited with the reward whenever it is received from environment. While in multi-step environments, the classifiers in previous iteration are credited with a payoff amount, which is devoted by the classifiers in the current iteration. The update policy is the same in both cases, apart from the ActionSet within which the update occurs and the source of the update value, reward or payoff.

The updated parameters in classifier include Prediction, Error, Accuracy, and Fitness values. In addition, other parameters, such as the ActionSetEstimate, Experience and the GATrigger, are updated for Induction Algorithms in Sect. 2.2.4.

Single-Step Problems

For a given reward, the update for all classifiers in ActionSet is as follows. The update of Prediction, Error, and fitness depends on the standard Widroff-Hoff delta rule as Eq.(2–5).

$$\nu_j \leftarrow \nu_j + \beta(V - \nu_j) \tag{2–5}$$

In detail, the prediction p is updated using the Widrow-Hoff delta rule with learning rate $\beta : (0 < \beta < 1)$:

$$p = p + \beta(reward - p) \tag{2–6}$$

The prediction error ε is updated with the formulary:

$$\varepsilon = \varepsilon + \beta(|reward - p| - \varepsilon) \tag{2–7}$$

The fitness update is slightly more complex. Initially, the prediction error is used to calculate the *accuracy* κ of each classifier as Eq.(2–8) when $\varepsilon > \varepsilon_0$, else $\kappa = 1$.

$$\kappa = \alpha * \left(\frac{\varepsilon_0}{\varepsilon}\right)^5 \tag{2–8}$$

Finally, the fitness are adjusted:

$$f = f + \beta * \left(\frac{\kappa * n}{\sum_{cl \in A} \kappa * n} - f\right) \tag{2–9}$$

The accuracy is computed directly from the prediction error ϵ parameter, and fitness is derived from the accuracy of all classifiers within the ActionSet and is therefore updated using the Widroff-Hoff delta rule.

The "experience" x is maintained by each classifier, which maintains a count of the number of times the classifier has occurred within an Action Set. The "ActionSetEstimate" e calculates the number of MicroClassifiers in all the ActionSet (n means the number of classifiers in a MacroClassifier). The "GAIteration" g is set with the present timer.

$$x = x + 1$$
$$e = \sum_{cl \in A} n$$

Multi-Step Problems

For a multi-step problem:

1. If a reward value is obtained from environment, invoke the single-step algorithm for the current ActionSet.

2. If this is not the first step of a trial, calculate the maximum prediction array in the current ActionSet as payment P, discount it using parameter γ, and then, apply it to classifiers in ActionSet of previous iteration.

3. Invoke the same algorithm as that of single-step algorithm, but replacing the reward with the discounted payment P, and the ActionSet is set with that of previous iteration.

The use of the maximum prediction array as the payoff ensures that the route to the highest rewarding state is selected.

More general classifiers will occur in ActionSet more often than specific classifiers. They will therefore be given more opportunity to duplicate or breed than an equally accurate

specific classifier. This mechanism will proliferate the general classifier, and drive out the equally accurate but less general classifiers from the population gradually, by combining with the deletion techniques used when new classifiers are introduced by the induction algorithm Sect. 2.2.4.

2.2.4 Induction Algorithm

The Induction Algorithm provided in XCS consists of covering operator and GA.

Covering Operator

The covering operator is invoked whenever no classifiers exist within the population that matches the input message. It creates a new classifier which condition is copied from the input message, but replaces selected bits with wild-card with generality probability, and action is set at random. Then set the initial parameter.

GA

The GA is triggered in explore trial. The classifiers involved in GA will be only those within the ActionSet. This means that the GA does not explore actions using the crossover operator, but rather explores conditions. Thus, the XCS is not seeking to create correct responses, but a complete concise mapping of state action payoff with maximally general accurate classifiers. The working schedule is as follows.

Step1: GATrigger.

GA is triggered when the interval from last occurrence of GA is greater than a const parameter θ. Here, we set θ to 25. The interval is calculated by the distance between the present timer and the averaged GAIteration of all classifiers in ActionSet.

Step2: Select Classifier with Roulette Wheel Policy.

Select two parents using Roulette Wheel Selection over the fitness values of the classifiers within ActionSet. The parents are selected without removal so that the same classifier may be chosen twice to allow the duplication of classifiers.

Step3. Crossover.

If the two parents differ, perform single-point crossover within the condition with the crossover probability, otherwise copy the parents to the children.

Step4. Perform mutation on both the condition and action.

Mutation within the condition operates at the level of the ternary digits. The ternary digits to be changed are chosen using the mutation probability. The specification of action mutation has been deliberately left as general.

Step5: Initialize child classifier.

If the child classifier is performed with crossover, set its initial prediction to the mean of parental prediction; otherwise, copy from that of its parent. Set the prediction error to one quarter of the population mean error, fitness to one tenth of the population mean fitness, ActionSetEstimate to the population mean, numerosity to 1, and experience to zero respectively.

Step6. Insert child into population.

For a new classifier, no matter it is created by coving operator or by GA, it must be inserted into the population. To do this, another classifier may need to be deleted if the population is already full.

Delete: Once a classifier is selected for deletion, the numerosity of the selected classifier is decreased by one and if the numerosity has become to zero, the classifier is removed from the population. Then it will create one free micro-classifier location within the population for the new classifier.

Here we use Roulette Wheel Selection to select the deleted classifier based on ActionSetEstimate. Because the ActionSetEstimate reflects the number of micro classifiers occurring in the ActionSet that a classifier occurs within, the larger ActionSet will be more likely to have a classifier selected for deletion.

Insert: A new classifier may be a duplicate of an existing classifier, or it may already be covered by a more general classifier that has already been evaluated as accurate. Then the numerosity and ActionSetEstimate of the duplicate classifier is increased by one and the new classifier is discarded.

Since more general classifiers will occur in more ActionSet, they will therefore be updated more regularly and therefore have more breeding opportunities. Then, more accurate general classifiers will increase, whilst less general accurate classifiers will gradually diminish.

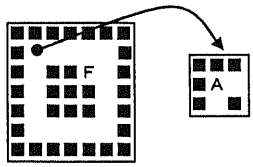


Figure 1: Woods1.

2.3 Application and Recent Work

2.3.1 Demonstration of XCS

Generally, to demonstrate the performance of Classifier System, the Multiplexer [24] and Grid-Maze were used as single-step problem and multi-step problem respectively. In this section, we mainly focus on multi-step.

For simplification, we use Woods1 as the left part in Figs. 1 for test, which is one of the usually used test environments to evaluate the Classifier System. The agent is placed on a random free starting position and allowed to move in the eight directions. An attempt to move into a position of Wall '■' will result it remaining in the original position. An attempt to move onto a position food 'F' will result a success with a reward value, and then start the next episode.

The perception of agent consists of the encoded contents of its eight surrounding positions. We encode each grid position as: Wall '111', Free '000', Food '001'. Then the perception is encoded in a 24-bits string. For example, if the agent is placed in the top left corner in Woods1, it will perceive the environment as "111,111,000,111,000,111,111,111". Here, we set its perception from North direction, and running in a clockwise direction. Action is encoded using integer value. For the agent in this instance, its optimal action is "go east", then we set it as 2. The other parameters are set as in Table 1.

The performance of XCS is defined by number of steps taken to achieve a reward within each trial. The measure of System Error and Population Size is kept as additional performance. System Error is the absolute difference between the average Prediction value for the chosen action and the actual payoff that the action results. Population Size is the number of Classifiers within the population. The smaller the Population Size, the higher compacted search space has been obtained. Here ten experimental results have been averaged.

Those trials that cause a step immediately to a food grid position will receive an immediate

Table 1: Parameter Value.

paraName	value
N(popSize)	800
θ(GA experience)	25
χ(crossover probability)	0.8
μ(mutation probability)	0.01
β(learning rate)	0.2
γ(discount rate)	0.71
ε_0(minimum error)	0.01
P_i(initial prediction)	10.0
f_i(initial fitness)	0.01
ϵ_i(initial error)	0.0
P_x(explortion/exploitation)	0.5
P_*(# probability)	0.66

reward of 1000. While those that are two grid positions away will receive the discounted maximum prediction of the directly rewarded classifiers, with a discount value γ. One further step away will obtain a less discounted value.

The graph, shown in Figs. 2, demonstrates that this version of XCS is able to solve the Woods1 problem. The average number of steps to a reward from a random starting position is 3, and the System Error is near to 0. The Population Size grew to approximately 350, while the whole population size(micro-classifier) is 800. Apart from this, an analysis of the population showed that the accurate classifiers had converged to stable payoff predictions of 1000, 710 ($\gamma * 1000$), and 504 ($\gamma * 710$), where there was no error, as predicted. The results of the current implementation give a confidence that this implementation of XCS produces a similar outcome as we have designed, and it is comparable with the previous LCS in multi-step problem domains. It is therefore concluded that the XCS is a sufficient implementation for further studies.

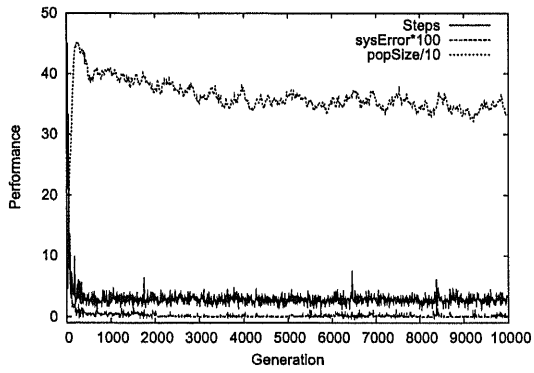


Figure 2: Performance of XCS applied on Woods1.

2.3.2 Recent Work about Classifier System

Since LCS was proposed in 1978, XCS was first proposed in 1995 and identified in its present form in 1996, they have been widely expanded from binary value [10, 11, 12] to real value environment [13], and applied in various fields, such as data mining [14], stock market analysis [15, 16], and so on. This section surveys the major devotion on Classifier System, and expands a list of areas of their work for further investigation.

(1) DataMining; Real-World Application of XCS

Besides the research on virtual-world problems, [26] describes an application of XCS to a Data Mining task, based on Classifier System and XCS. Given a set of attributes from a data set, XCS would be applied to learn the relationship among the plural independent attributes, and analyze the changing regularity to that of one single dependant attribute. Using a well-known sample data set from the UCI Data set Repository, this work showed that XCS was able to produce a better classification performance. It also confirmed that XCS can converge in single-step problem not only the Multiplexer.

On the other hand, LCS is also pursued on many real-world applications [27], divided into three main sections: data mining, modeling and optimization, and control.

(2) Extensions of XCS to Real Value Environment

In [28], Wilson modified XCS so that the condition consisted of a vector of intervals, where each interval was represented by a pair of real number values. One represents the central point and another represents the spread on either side of the central point. In order to assess the ability of XCS for real value environment, he applied this vector

to six-multiplexer test, and six real number intervals between 0.0 and 1.0 were provided in the condition. The result showed that the performance close to 1.0, and the classifiers in the population with the highest fitness, but with low numerousity for each classifier. This is mainly because the real vector results a huge search space, and it prevents precise classifier required for subsumption.

In [13], a detail analysis of the real value XCS has been pursued. The further experiments showed that mixed binary/real number conditions are also able to find and establish the optimal sub-populations.

(3) Applying Memory for Non-Markov Problem

In [29], a ZCS with bit memory has been proposed to complement the lack of a message list. Their work demonstrated that ZCS was able to utilize the memory bits to solve problems in Non-Markov environments, although the learning became unstable when the number of bits was expanded. In [30], Lanzi utilized the idea of memory bits on XCS (XCSM) to provide a solution to the "aliasing problem". As for the detail, we will introduce in Chapter 3.

2.3.3 Future Work

From these analysis and experiments, we concluded that the XCS is a satisfactory platform for the further research, as a reliable robust Machine Learning architecture. And as we have demonstrated, considerable progress has been achieved within a short time. Clearly there is much more work to be done, and some further work is possible on the use of internal state, or on hierarchical invocation of classifiers.

In addition, the RL community has gained considerable benefits, and similar benefits would be seen for XCS investigations. Then a possible alternative avenue would be to apply RL approaches to XCS.

It is clear that the real world provides many opportunities for research work, and it will attract more workers to use the Classifier System.

Chapter 3

Adaptive XCSM for Aliasing Problems

3.1 Introduction

Due to the limitation of agent's sensor, it maybe result in an agent failing to obtain enough information to distinguish between two different situations, which appear identical to the agent, but require two different actions to behave. Such an environment is said to be Non-Markov and the agent is suffering from a perceptual aliasing problem [31], which disturbs the agent's learning capability seriously. To overcome this problem, generally, we endeavor to expand agent's sensation to convert the Non-Markov environment into a Markov one.

In [24], an improved classifier system, XCS has been introduced, which is based on the accuracy of the classifier's payoff prediction instead of the prediction itself. It has been shown to reach optimal performance in Markov problem, but failure in Non-Markov problem.

Based on ZCS [23] and ZCS with a temporary memory [29], XCSM [30] has been proposed by adding bit-register memory to XCS. For the pre-assigned fixed-length memory, there are two flaws. One is how to define the fixed length. Generally, we set it by counting the aliasing positions in a given environment. For a simple map, it is easy. But for a harder one, it is not always so exactly to realize. The second problem is that, assume that we have gotten a suitable memory length for the maximum aliasing positions, but for any other positions, it is not necessary to use so much long memory. This results in the space waste.

Therefore, we proposes an adaptive XCSM (AXCSM), in which only the maximum memory length is set beforehand, and the appropriate length for each classifier varies from 0 to the

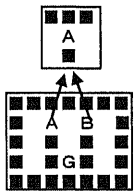


Figure 3: Woods101.

maximum. AXCSM is realized through a hierarchical structure. The first hierarchy is to learn a suitable memory length for any given position, using general XCS. In the second hierarchy, action is obtained by XCSM. Experiment is pursued to verify the validity of AXCSM.

The remainder of the chapter is organized as follows. Section 3.2 presents the perceptual aliasing problem and XCSM. Section 3.3 describes the adaptive memory implementation. Experiments are presented and analyzed in Sect. 3.4. In Sect. 3.5, the future work has been prospected.

3.2 Related Work

3.2.1 Perceptual Aliasing Problem

A typical perceptual aliasing problem is shown in Figs. 3. In this thesis, we indicate that the agent perceives environment by means of Boolean sensors that report the contents of its eight surrounding squares. And the agent moves into eight adjacent directions, encoded using integer value. The learning objective is to find a shortest path to goal position from random start.

To simplify, the encoding for each grid position is listed in Table 2. The agent perceives from North direction, and running in a clockwise direction. And the action is encoding as in Table 3.

Table 2: Perception encoding for environment.

Symbol	Meaning	Encoding
Null	Blank	000
'■'	Wall	111
G	Goal	001

Table 3: Action encoding for environment.

Action	Encoding
North	0
NorthEast	1
East	2
SouthEast	3
South	4
SouthWest	5
West	6
NorthWest	7

For example, if the agent is placed in position marked with 'A' as in Figs. 3, it will perceive the environment as "111,111,000,000,111,000,000,111", and its optimal action is "go east", which is marked with integer value 2.

As for the Figs. 3, we can see that the agent has an identical perception for two distinct locations 'A' and 'B'. To reach goal state 'G', for position 'A', the optimal action is "go southeast"; while for position 'B', it is "go southwest". The agent couldn't distinguish the two locations, simply basing on its sensation of the current position in environment. Learning for this kind of perceptual aliasing problem is necessary and urgent.

3.2.2 XCSM

We have introduced the principle of XCS and applied it in a traditional Maze test problem in Chapter 2. With the same setting, we applied it on Woods101 in Figs. 3, and found that XCS could not obtain the optimal solution to this environment, because of its Non-Markov property. To behavior optimally in Woods101, the agent needs some form of memory to cope with the lack of information provided by its sensors. We can follow two approaches to add memory to the agent. One is to use a "history window" to store its previous inputs. However, in this case, the agent's input space grows exponentially in the size of the history window. The other is just using one or a few stored bits, instead of a list of messages, to make decisions based on past information. In the later, past experience is not stored explicitly as in the history-based approach, but the agent must learn to use the memory to solve perceptual aliasing. For example, let's consider an agent with a one bit memory register

to solve Woods101 optimally. If the agent is in the left side of the maze, it sets register to 0; if the agent is in the right side of the maze, it sets register to 1. When entering an aliased position, the agent selects the action to perform depending on the value of the register: if it contains 0, the agent performs "go southeast"; if it contains 1, the agent performs "go southwest".

Based on this analysis, Lanzi propose XCSM in [30] for Non-Markov environment, adding a memory of bit-string on XCS. In XCSM, an additional register with m bits is added to XCS architecture. And the classifier is extended with an *internal condition* and an *internal action* from the condition and action of XCS, respectively, to *sense* and $modify$ the contents of the additional register. *internal condition* and *internal action* consist of m bits of characters in the ternary alphabet {0,1,#}. For *internal condition*, the symbols retain the same meaning as they have for the external condition, but they are matched against the corresponding bits of the register. For *internal action*, '0' and '1' set the corresponding bit of the additional register to '0' and '1', respectively; '#' leaves the corresponding bits unmodified.

Then, the classifier in XCSM is defined as follows. Here, the *classifierXCS* is illustrated in Sect. 2.2.2.

$$
\begin{aligned}
classifierXCSM :: & \\
subclassifier &: classifierXCS \\
inCon &: internal\,condition \\
inAct &: internal\,action
\end{aligned}
$$

$$
\begin{aligned}
internal\,condition &= (< 0 > \mid < 1 > \mid < \# >)^* \\
internal\,action &= (< 0 > \mid < 1 > \mid < \# >)^*
\end{aligned}
$$

As for the learning procedure, XCSM works similar as XCS. At each trial, the register is initialized by setting all m bits to zero. The difference lies in that in XCSM, *internal condition* must also matches with register when building MatchSet, and each combination of an external and an internal action results in a distinct system prediction. The ActionSet is created, but all classifiers in it have the same external/internal action. The external action is set to the environment, while the internal one modifies the content of register. Those parameters update and GA operation work as in XCS.

The experiment has shown that the XCSM utilizes the memory bit to disambiguate the aliasing positions effectively. At the same time, we found that, for two aliasing positions, XCSM can achieve an optimal solution with one bit of memory; but for four aliasing positions, just two bits of memory is not enough. We must increase the memory length. However, the problem would be exacerbated as the number of internal bits increases.

Considering a limitation situation, when involving *internal condition* and *internal action* of m bits, the search space of internal memory will be a maximum of 3^m. With the m becomes larger, the search space will increase urgently. Therefore, adaptive XCSM (AXCSM) is proposed in this thesis. Our objective is to get a smaller classifier set, and a shorter memory for each single classifier as far as possible.

3.3 AXCSM

3.3.1 Architecture of AXCSM

In AXCSM, agent perceives its present position, and records its past experience by a register the same as that of XCSM. But additional parameter *memLength* and *inStrength, inValue, inNumerousity* have been added to classifier, and defined as below.

$$
\begin{aligned}
&classifierAXCSM ::\\
&\quad subclassifier2 : classifierXCSM\\
&\quad memLength : memorylength(Integer)\\
&\quad inS : inStrength\\
&\quad inN : inNumerousity\\
&\quad inV : inValue
\end{aligned}
$$

$$
\begin{aligned}
&inStrength ::\\
&\quad inP : inPrediciton\\
&\quad in\varepsilon : inError\\
&\quad in\kappa : inAccuracy\\
&\quad inF : inFitness
\end{aligned}
$$

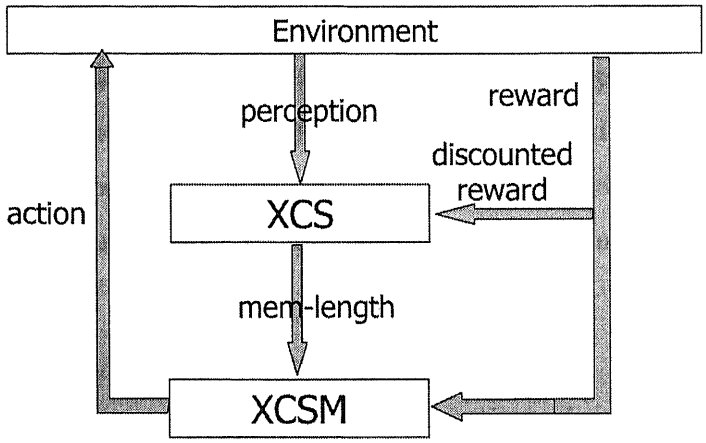


Figure 4: Framework of AXCSM.

$$inValue ::$$

$$inE : inActionSetEstimate$$

$$inX : inExperience$$

$$inG : inGAIteration$$

These parameters are defined the same meaning as those corresponding ones in XCS. While, the length of bit-string of *internal condition* and *internal action* are not fixed, but varying with *memLength* from 1 to the assigned maximum in advance.

The framework of AXCSM is outlined as Figs. 4.

Continue this loop until termination criteria is met.

(1) Hierarchy 1: get suitable memLength by general XCS.

In this heirarchy, for learning with XCS, set condition part as the pair of *external condition* and *internal condition*, and set action as *memLength*. The parameters *inStrength*, *inValue* and *inNumerousity* are appointed for getting an optimal action, which is defined as *memLength* here.

(a) Set starting position and Register content randomly.

(b) Generate inMatchSet.

Form inMatchSet with classifiers from PopulationSet, whose condition and *internal condition* matches with agent's perception and register respectively. Here, because the register is m bits length, and the length of *internal condition*

is variable, we just match them within effective range. If no matched classifier exists, a new classifier is created using the Covering Operator. The external and internal condition matches with agent's perception and content of register respectively. External and internal action are set randomly. While the internal condition and internal action are limited with the *memLength*, which is set randomly well.

(c) Get suitable action (memory length).

In an explore approach, we set the memLength randomly; otherwise, we calculate PredictionArray for each memory length as Eq.(3–1), and then set the maximum one as the optimal one. At last, those classifiers in inMatchSet, which have the same memLength with the optimal memory-length, are devoted to composing inActionSet.

$$predictionArray[memLength] = \frac{\sum_{cl \in inM} inP * inF}{\sum_{cl \in inM} inF} \tag{3–1}$$

(2) Hierarchy 2: get optimal action and *internal action* by XCSM.

(a) Get MatchSet.

The MatchSet is composed of all classifiers in inMatchSet, whose *memLength* equals to the selected memory length at the first hierarchy.

(b) Get action and *internal action*.

The combination of action and *internal action* is determined by the max PredictionArray. Those classifiers in MacthSet, which include the selected action and *internal action*, are identified to form ActionSet.

(c) Execute action and *internal action*.

Pursue action to environment, and update agent's perception. At the same time, the *internal action* is used to update content of Register.

(d) Pursue Learning Algorithm and Induction Algorithm.

When receiving a reward from environment, we pursue Learning Algorithm as general XCS for multi-step problem, and involve the Induction Algorithm.

(3) Pursue Learning Algorithm and Induction Algorithm on the first hierarchy.

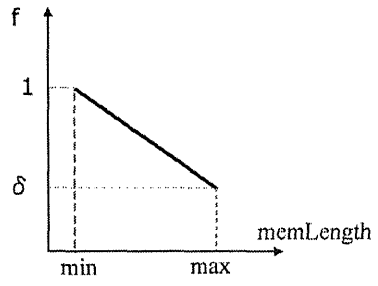


Figure 5: Discounted reward of AXCSM.

To update $inValue, inStrength$ and $inNumerousity$ with learning algorithm, the discounted reward or payoff, obtained from the second hierarchy, has been involved. This will be illustrated in Sect. 3.3.2.

3.3.2 Discounted Reward Strategy

As mentioned in the previous section, the parameters in *inMatchSet* will be updated with discounted reward or payoff, according to Eq.(3–2) and Figs. 5. Here, x means *memLength*.

$$r = payoff * f(x) = payoff * \left(\frac{(\delta - 1) * x}{max - min} + \frac{max - min * \delta}{max - min}\right) \tag{3–2}$$

In Eq.(3–2), *payoff* is reward value received from environment, or payoff obtained from the classifiers of next iteration. *minLength* and *maxLength* is the pre-set minimum and maximum memory length. δ is a constant ranging from 0 to 1(we set it 0.5 here). When receiving a payoff, if the memory length is *minLength* bits the whole value is paid; if the memory length is *maxLength* bits, the allocated payoff will be discounted according to δ. The really received reward value decreases with *memLength* linearly.

3.3.3 Delete Seldom-Accessed and Low-Fitness Classifiers

By analyzing the PopulationSet in XCSM, we found that there exist some classifiers, which are generated at the beginning of the learning process, and are seldom accessed, although we have involved the Induction Algorithm. Then we delete those classifiers that are satisfied with Eq.(3–3).

$$\frac{x}{(t - g)} * \frac{(inX)}{(t - inG)} < p1 * p2 \tag{3–3}$$

Here, for each classifier, t is the present timer value; x is the accessed times, g is last accessed time in ActionSet of second hierarchy by XCSM; inX is accessed times, inG is

last accessed time in inActionSet at the first hierarchy by XCS; $p1$ and $p2$ are discounted coefficient, set 0.01 here.

Meanwhile, there are also some classifiers with low fitness that contributes less to the learning procedure. They are deleted if Eq.(3–4) is satisfied. α is set as 0.05. Therefore, if fitness of a classifier is smaller than the probability of the average fitness, it will be deleted.

$$f < \alpha * \frac{\sum_{cl \in P}(f * num)}{\sum_{CS}(num)} \tag{3–4}$$

3.4 Experiment and Result

3.4.1 Experimental Design

The first experiment entails the implementation of a simple environment Maze3, shown as in Figs. 6. The agent perceives eight squares adjacent to itself as WALL ('■'), Free (Blank) or Goal ('G'), by means of three Boolean sensors for each cell. The action consists 8 directions movement (North, NorthEast, East, SouthEast, South, SouthWest, West, NorthWest). The learning objective is to get the shortest path to the goal position.

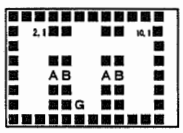


Figure 6: Environment Maze3(marked with two aliasing states and start coordinates).

In Maze3, the two positions marked with 'A' are aliasing ones. The agent perceives identical sensation from these two different positions, but it needs two different optimal actions to reach goal 'G' within the shortest steps. Similarly, the agent perceives a single sensation for the two positions marked with 'B', and needs two different actions as well. To recognize these two aliasing states, we let the agent start from two corner positions (2,1) and (10,1) randomly.

To explain clearly, we refer AXCSM which has m-bit minimum and n-bit maximum memory length as AXCSM-m-n. If the discussion is independent of the memory length, we refer to AXCSM simply.

During the learning procedure, action selection strategy alternatives between explore and exploit policy. In the final 2000 trials, the explore strategy is turned off, and only the exploit

one is used to monitor the learning result. Other parameters are set the same as those of [30]. The learning strategy for each hierarchy is summarized in Table 4 in detail.

Table 4: Learning strategy during the learning procedure.

		Explore	Exploit
XCS	memLength	randomly	determinely
XCSM	external action	randomly	determinely
XCSM	interanl action	determinely	determinely

For the first hierarchy, XCS, the memory length is set randomly in explore procedure, and is set determinately in exploit. At the second hierarchy, XCSM, the external action is set randomly and determinely in explore and exploit procedure respectively. However, the internal action is determined by the maximum PredictionArray in both explore and exploit procedure.

The agent's performance is judged by three aspects, averaged steps to goal position in every 100 trials; averaged number of classifiers in PopulationSet for every 100 problems, and the memory length for all classifiers in the PopulationSet after one experiment. All the performance is compared with that of XCSM.

3.4.2 Experiment on Maze3

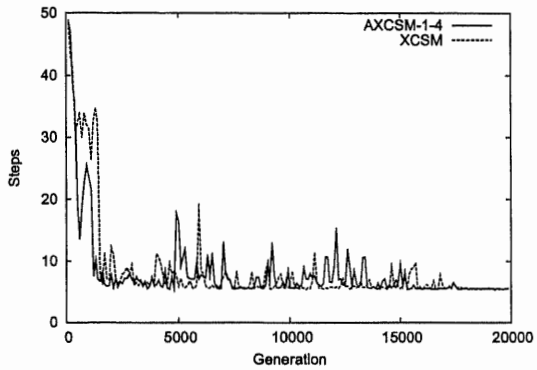
If the maximum memory length is set by 2 bits or 3 bits, the effectiveness of AXCSM is not so prominently. Since even if all the classifiers are set with the same memory length as the maximum length, the internal space is the same as that in XCSM, and is not so enormous. However if we set a longer memory length, the result is apparent. Here we set AXCSM as AXCSM-1-4, the max population as 4000, and 20000 trials have been pursued.

The result for AXCSM-1-4 is shown in Figs. 7, and XCSM is set with 4 bits memory length.

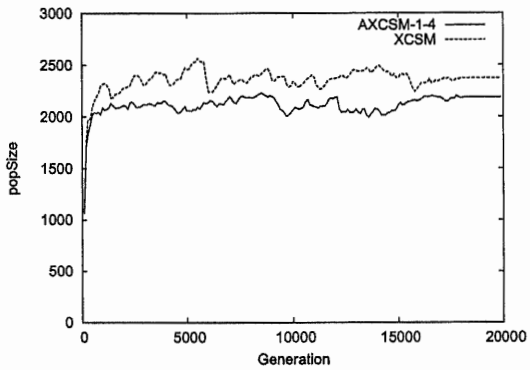
Figure 7(a) shows the average steps to the goal position for both AXCS-1-4 and XCSM. We can see that both of them converge to the same optimum, but the proposal converges more slowly. The reason is that at the beginning stage, the agent must try to get suitable memory length for each classifier, so that it could not contribute to a suitable action.

Figure 7(b) shows the macro-classifiers number fluctuating with the learning procedure.

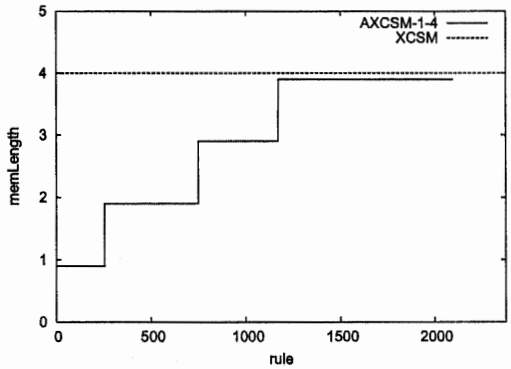
The macro-classifiers number in AXCSM-1-4 is less than that of XCSM. Then we can conclude that the proposal searches in a compacted space.



(a) Average steps to goal position.



(b) Size of PopulationSet.



(c) Variable memory length for the final PopulationSet.

Figure 7: Performance of AXCSM-1-4 on Maze3, compared with XCSM.

Figure 7(c) shows the memory length of all classifiers in the final PopulationSet after 20000 trials. It is the highlight of this proposal. We have mentioned in Sect.3.3.2 that we allocate a discounted reward to classifier according to its different memory length. The classifiers with shorter memory length will be more valuable and will survive with higher opportunity, even if they receive the same reward from environment, and adversely, the classifiers with longer memory length will be eliminated more easily. In Figs 7(c), the horizontal axis is the classifier's serial number, and the vertical axis is memory length for each classifier. We summary it in Table 5.

Firstly, the curve of XCSM ends at about 2373 in horizontal axis. This means that there

Table 5: Analysis of PopulationSet on Maze3.

	populationSize	ave-memLength
XCSM	2373	4 bits
AXCSM-1-4	2091	2.963 bits

are 2373 macro-classifiers in the final population. However, in AXCSM-1-4, the number is 2091. The proposal has less number of classifiers than that of XCSM. The result is consistent with that of Figs. 7(b).

Secondly, in XCSM all the classifiers have the same memory length which is 4 bits. While in AXCSM-1-4, the classifiers consist of hierarchical distribution, with memory length ranging from one bit to four bits, and average to 2.963 bits. Thus the whole memory space becomes more contractible.

3.4.3 Experiment on Woods102

To further verify the proposal, we apply it to another more complex environment Woods102 shown in Figs.8(a). In Woods102, the positions marked with 'A' are aliasing positions need four different actions, and positions marked with 'B' need two actions. In Figs.8 (b), the data pairs denote the coordinates in a two dimensional space.

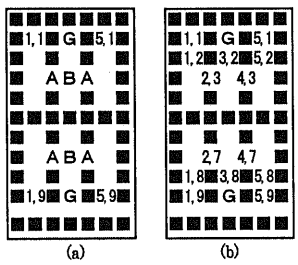


Figure 8: Environment Woods102, (a) aliasing states and start coordinates, (b) coordinates.

Here we assume that the AXCSM is set as AXCSM-1-8, and the settings are the same as those of Sect.3.4.1, except that the max population is 6000. We suppose that agent starts from four corner positions (1,1), (1,9), (5,1), (5,9) randomly.

The result analysis is similar to those of Figs. 7 (not shown here). We show the agent's trial track in Figs.9, using the population obtained after one experiment by AXCSM-1-8.

Figure 9(a) describes the agent's trial track, which starts from four corner positions respec-

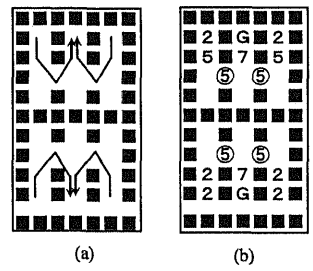


Figure 9: Agent's track on Woods102 using AXCSM-1-8.

tively, and reaches to the goal position within minimal steps successfully.

In Figs.9(b), the integer value is the memory length of each position on the route to the goal. This means that when agent reaches to this position, the agent needs this number of bits memory to remember its past experience. For example, in order to recognize the four aliasing positions marked with circle, it needs 5 bits register; to recognize the positions (3,2) and (3,8), 7 bits register is necessary. This means that only those classifiers, whose length of *internal condition* and *internal action* are limited with the assigned length can be used to determine the suitable action. The shorter the memory is, the more compacted space is obtained.

However, as seen in Figs.8(b), the position (1,1) and (5,1) are not aliasing positions, and thus we don't need any additional memory to recognize them theoretically. The same situation occurs at position (3,2) and (3,8). However, as seen in Figs.9(b), memory space has been involved to recognize them respectively, which results in the waste of the search space. To analyze the changing mechanism of the memory length, we set it from 0 bit, and observe it further.

3.4.4 Further Experiment on Woods102

Based on the analysis in Sect. 3.4.3, we introduce an extension to the initial AXCSM, in which the memory length varies from 0 bit to maximum, instead of from 1 bit. Now, we apply AXCSM-0-8 to Woods102 with the same setting as AXCSM-1-8. The result of both AXCSM-0-8 and AXCSM-1-8 are shown in Figs.10 for comparision.

Figure 10(a) and Figs.10(b) show that AXCSM-0-8 and AXCSM-1-8 have similar performance on Woods102 in terms of the converge speed and the macro-classifiers number fluctuation. The memory length of the final population is summarized in Figs.10(c) and Table 6.

We can observe that the improvement is prominent.

Table 6: Analysis of PopulationSet on Woods102.

	populationSize	ave-memLength
AXCSM-1-8	3273	5.125 bits
AXCSM-0-8	2829	3.95 bits

Comparing the agent's trial track on Woods102 in Figs.9 and Figs.11, we found that the agent has the identical path to the goal position within minimum steps. However, the memory length on its route is different. As seen in Figs.11(b), in order to recognize the aliasing position (2,3), (2,7), (4,3), (4,7), we need 5 bits internal memory, the same as that of Figs. 9(b). Nevertheless, to recognize the non-aliasing position (3,2) and (3,8), (1,1) and (5,1), one bit internal memory is enough, which is greatly shorter than that of Figs. 9(b). This result shows that AXCSM-0-8 is more effective than AXCSM-1-8, although we still can not recognize all the non-aliasing positions with optimal policy by now.

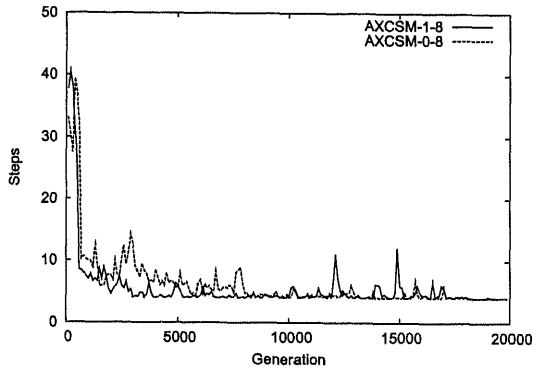
3.4.5 Analysis

To verify the variable memory length being more reliable, we fetch the populationSet during the learning procedure, and apply it as knowledge-base to guide agent's action. In this experiment, we fetch populationSet 5 times at generation 5000, 10000, 15000, 18000, 20000 during the 20000 generations learning procedure, and then apply them as knowledge-base to direct agent's action respectively. For each population, agent starts from four corner positions. The average memory length for each trial is listed in Figs. 12. The averaged memory length decreased gradually with the learning procedure going on.

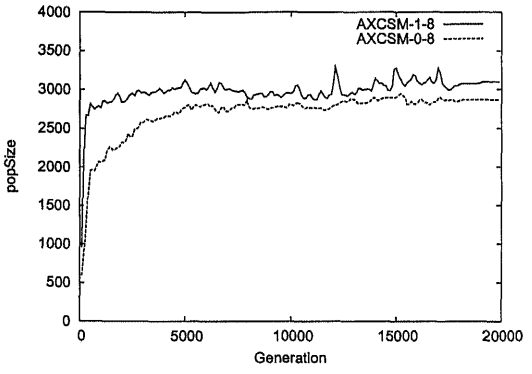
Furthermore, we show the trial result based on populationSet at generation 10000, 18000 and 20000. Their register content is illustrated respectively at Table 7. First, we found that the three settings converged to optimal routine within their trials, the same as that of XCSM. Second, at each trial, the agent recognizes aliasing positions by involving different register content and valid register length. For example, at generation 10000, the different registers at (2,3), (2,7), (4,3), (4,7) are successful to set optimal action, and the same efficiency occurs to generation 18000 and 20000. With the learning proceeding, the length of valid register decreases as we have expected(shown in the last row). Compared with that, for XCSM, the register is a constant of eight bits. The efficiency of proposed AXCSM is apparent.

Table 7: Variable memory length and register content during learning on Woods102 using AXCSM-0-8.

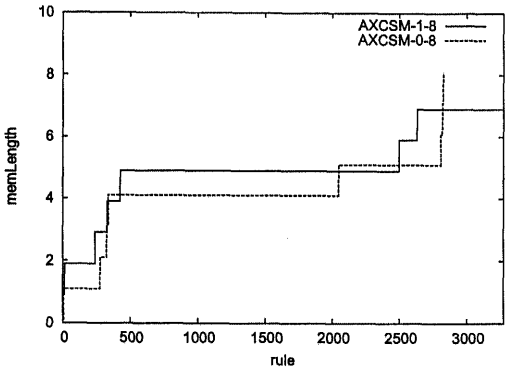
gen = 10000				gen = 18000				gen = 20000		
pos	mLen	reg		pos	mLen	reg		pos	mLen	reg
(1 1)	4	0000		(1 1)	1	0		(1 1)	1	0
(1 2)	5	10110		(1 2)	4	0000		(1 2)	1	0
(2 3)	7	1100000		(2 3)	5	10000		(2 3)	5	10000
(3 2)	5	10010		(3 2)	1	1		(3 2)	1	0
(1 9)	3	000		(1 9)	4	0000		(1 9)	1	0
(1 8)	5	01100		(1 8)	4	0000		(1 8)	4	0000
(2 7)	5	11011		(2 7)	5	01010		(2 7)	5	01010
(3 8)	4	0011		(3 8)	1	1		(3 8)	1	0
(5 1)	4	0000		(5 1)	1	0		(5 1)	1	0
(5 2)	5	10110		(5 2)	4	0000		(5 2)	4	0000
(4 3)	5	00100		(4 3)	4	0010		(4 3)	5	00000
(3 2)	5	00100		(3 2)	1	0		(3 2)	1	0
(5 9)	3	000		(5 9)	1	0		(5 9)	1	0
(5 8)	5	01100		(5 8)	4	0000		(5 8)	4	0000
(4 7)	5	01001		(4 7)	4	0110		(4 7)	4	0110
(3 7)	4	0101		(3 6)	1	0		(3 8)	1	0
(3 8)	4	0110		(3 7)	4	0100				
				(3 8)	1	0				
average memory Length (bits)										
4.59				2.81				2.5		



(a) Average steps to goal state.



(b) Size of Population-Set.



(c) Variable memory length for the final Population-Set.

Figure 10: Performance of AXCSM-0-8 on Woods102, compared with AXCSM-1-8.

3.5 Conclusion

We propose an adaptive XCSM (AXCSM) to more complex Non-Markov environment. The classifier in XCSM has a fixed length of memory to record its past experience. However, with the fixed length becoming longer, the search space expands as well. Furthermore, we involved a variable memory length to XCSM (AXCSM), ranging from 1 bit or 0 bit to the maximum length which is defined beforehand. Through experiment, we can observe that by using this proposal, smaller population can be obtained and the memory length for each classifier can also be decreased.

Based on the above analysis, we can conclude that the AXCSM outperforms XCSM, especially for complicated maze. However, on one hand, we still need to further analyze the

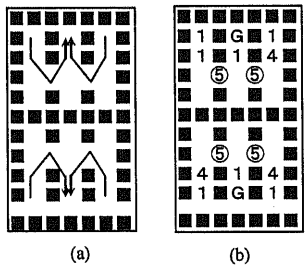


Figure 11: Agent's track on Woods102 using AXCSM-0-8.

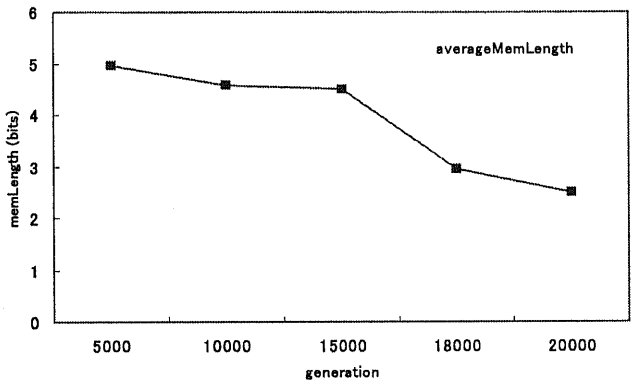


Figure 12: Average memory length during learning on Woods102 using AXCSM-0-8.

changing mechanism of the memory length and its content. On the other hand, we can not foresee how the AXCSM will behave in real-value environment. This will be our concern in the future.

Chapter 4

Hierarchical XCS and Its Application to Stock Market Forecasting

4.1 Introduction

In the past decades, the results of various quantitative forecasting techniques have served as important tool for strategic decisions or pre-analysis of time-series data in a number of fields, such as the stock market, weather forecasting and industrial flowing control. The data consists of a single series, or multiple variables. The objective of forecasting is to find some kind of nonlinear regularity among these variables involved, and to give a good explanation.

In this thesis, we mainly focus on the forecasting of financial market as a concentricity of mankind. However, modeling real financial market is not an easy task for financial trading and investment management. The financial market is influenced by a significant number of elements whose relationship is complicated. This makes the problem of economic forecasting very hard, and no actual knowledge can be used to judge what is the right investment approach. Then we will review these situations in the following sections.

4.1.1 Classical Economic Theories

The Efficient Markets Hypothesis (EMH) is one of the milestone of current financial theory. In this market, mathematical models of price movements are based on the assumptions of rational traders, equilibrium analysis and all relevant information possessed by the traders. Any deviations from these idealized condition are considered to be exogenous effects or un-

certainty. The hypothesis first appeared in the PhD dissertation of Eugene F. Fama at the University of Chicago, under the title "The Behavior of Stock Market Prices". The main idea of EMH is that in an efficient market the actual price is the good estimate of its intrinsic value. The stock market prices always reflect all information available to traders. The past information can not help predicting future prices, and the markets are assumed to be free of internal dynamics of their own.

The EMH also composes of three hierarchies with the first one being the "Weak" form. The "Weak" form asserts that in a market, it is impossible to predict the future price on the basis of its past price because all past market prices and data are fully reflected in prices, so the technical analysis of past prices is useless. The second one, "Semi-strong" form, refers to the market reaction based on public information such as news announcements, annual reports, etc. However, we still don't have a precise answer to what should be considered public information. A common guess is that the easier it has been obtained, the more likely it is to have already been traded upon. Therefore in this point of view, it is impossible to predict on the basis of publicly available fundamental information. Finally, the third one, "Strong" efficiency, analyzes whether investors have private information to take advantage of. The private information includes internal information, such as a personal note regarding a major financial decision which would have an impact in the stock price. However, most people don't believe that the market is strong-form efficient.

EMH became the dominant paradigm used by economists to explain the financial markets. As an opposite to the EMH, in [32], Andrew W. Lo and A. Craig MacKinlay provided an important evidence showing that financial market is not completely random. In this volume, they found out that predictable components did exist in recent stock and bond returns. By looking at a given historical sequence, it is clear that price tends to move in one direction for a long time. Another idea against EMH is that "the more people share a belief, the more that the belief is likely to be true." Therefore, people transact following the price of each day, and they will influence price movements. All of these results agree that the market returns are predictable, although the predictability varies over time. In any case, most stock market investors seem convinced that they can predict stock price trends.

4.1.2 Non-Classical Economics Theories

As analyzed in Sect. 4.1.1, the prediction of financial market as a time series becomes feasible to some extent. In the last decade, a number of different methods have been applied in order to predict stock market movement. These methods can be classified into categories of technical analysis methods, fundamental analysis methods, traditional statistics method, and intelligent method. Technical analysts, know as chartists, attempt to predict the market by tracing patterns from charts of the historic data of the market as described by Malkle [32] for example. Since this method may depend on psychological factors and technical, occupational knowledge, it is mainly used by experienced financial professionals and financial groups who have vast special knowledge. Fundamental analysts study the intrinsic value of a stock. It is helpful to predict the market on a long-term basis. The traditional statistics methods [33], such as linear auto-regressive analysis models and principal component analysis models, create linear prediction models to trace patterns of historical data. Finally, a number of methods have been developed in the intelligent method category, including Evolutionary Computation (EC), Machine Learning, as well as Neural Networks (NN), etc.

Here, we focus on the Intelligent Method, and, in particular, mainly on EC and Machine Learning. As effective learning algorithms, Artificial Neural Networks have been widely used to deal with stock market forecasting [9, 34], and are able to identify highly non-linear models. However, the main disadvantage of these methods is the absence of an internal memory, which makes it difficult to capture the dynamics of large-scale time series data. Although recurrent ANNs have been developed employing some form of internal memory [35], several problems occurred in the application of these methods. The bottle-neck problem is that local over-fitting is very likely to happen. If this problem does occur, the model is only appropriate for a specific period. For a large number of data, it is difficult to get optimal results. Moreover, the limits of readability of NN make it difficult to be analyzed because of its "black box" property, while the high CPU requirements of NN computational implementations are also well know.

Aside from NN methods, GA is another popular algorithm used in prediction fields [9, 36, 37]. These evolution inspired computational methods have received widespread use in financial time series forecasting.

In order to analyze the stock market efficiently, hierarchical learning systems, such as NXCS [15] and Hybrid Intelligent system [38], are often constructed by combining several

single algorithm together, and obtain better performance than those systems which use one single algorithm.

In [15], a mixture of hybrid expert consists of a genetic classifier and an associated artificial neural network. The former is used to find quasi-stationary regimes with the financial data series, and the latter is assigned the task of making predictions on the market changing trend of the next day. In [38], it deals with the hybridized techniques used for stock market forecasting and market trend analysis, by making use of a neural network for the next day stock forecasting and a new neuro-fuzzy system for analyzing the trend of the predicted stock values. In this system, a neural network is applied on the stock forecasting. After that, the deviation of the predicted value from the required value as a fuzzy variable and used a fuzzy inference system to account for the uncertainty.

Except for the methods involving one identical agent, as mentioned above, many other work has been focused on Multi-Agent system, which mainly constructs an artificial stock market to simulate the real market, to analyze its mutual similarity and internal regularity [39, 40], or to simulate the stock market to acquire the maximum profit by transacting with market [41, 42, 43]. All of these methods involve a Multi-Agent structure to analyze the internal dynamics of financial markets. Analysis of these agent based systems has shown that simulation of the real market can be used to explain the working of financial markets. Hence, it becomes possible to study price dynamics in a more diverse environment that can be closer to reality, and then analyze its properties.

For all the proposals referred above, they need to predict the trend of future price change by obtaining either a buy or a sell signal, based on the historical data of given market. Now we pay great attention to the changing direction forecasting of the next time, rise or fall. Based on the forecasted changing direction, rise or fall, we can give an indication to the next investment, sell or buy, to get maximum profit. In this paper, we proposed a Hierarchical XCS (HXCS), which integrates XCS [24], an GA-based RL algorithm. The HXCS is used to combine the knowledge of plural agents to remedy the insufficiency of one single agent. In the higher levels of the hierarchy, RL is used to determine how to shift among those local models for a changing trend. In the lower levels of the hierarchy, an agent is trained by the XCS method to learn and forecast.

The remainder of the paper is organized as follows. Section 4.2 describes the hierarchical learning architecture proposed in this thesis. Experiments are presented and analyzed in

Sect. 4.3, Sect. 4.4 and Sect. 4.5. Section 4.6 outlines a conclusion, and discusses the future work.

4.2 Hierarchical XCS

4.2.1 Classifier Definition

Generally, the most basic way of preprocessing data is by simple moving average. The moving average is calculated by taking an average of the last L values. The parameter L represents the period length for moving average. This kind of simple moving average works well in simulating and analyzing the financial markets. However, it is difficult to define the moving average interval L, and difficult to predict whether an agent can recognize real changes pattern effectively.

Moreover, it is difficult to predict all situations correctly by means of a single agent. It is, however, more possible to predict certain special situations, for example, an agent may be successful in situation A, but failure in situation B. We therefore use multiple agents here. Each agent concentrates on a local changing area. By the cooperation of all these agents, a complex task can be accomplished to a higher degree of accuracy.

Here, we make use of heterogeneous groups. In each group, there are several agents with the same learning strategy. Then for all agents, they may be homogeneous or heterogeneous. Each agent recognizes the environment through its own vision window, and learns by the XCS method. We now define the structure of each agent.

(1) Condition definition for heterogeneous groups.

We introduce two types of groups in this module, naming them Group1 and Group2. In order to set the agent's perception with a bit string, two steps are taken.

Step1: The agent perceives a series of moving average values.

Figure 13 shows the perception of agents in Group1 and Group2 at trading time t respectively. $MA_{t,m}$ means the average value from time t back to time t-m. In Figs. 13(a), the agent recognizes 24 moving average successively, with an interval length of 20. In Figs. 13(b), the agent recognizes its vision hierarchically. First, it gets 6 moving average, all with an interval length of 10. The 6 moving averages are calculated using continuous source data but without duplication. The agent then gets a further 12 moving average from time t with an interval

length of 5. These 12 average values are also calculated using continuous and no-duplicated data.

Step2: The agents' real value perception is transferred into a bit string for the purpose of learning by XCS.

In Fig. 13, if the inequality is satisfied, the corresponding bit value is set as '1', otherwise, it is set as '0'.

At this stage, we have thus successfully set an agent's perception formulated as bit-string. The agent in Group1 perceives 24 bits, while the agent in Group2 perceives 18 bits.

Bit	Representation
1	$MA_{t-24,20} < MA_{t-23,20}$
2	$MA_{t-23,20} < MA_{t-22,20}$
...	...
i	$MA_{t-i+1,20} < MA_{t-i,20}$
...	...
24	$MA_{t-1,20} < MA_{t,20}$

(a) Group1.

Bit	Representation
1	$MA_{t-59,10} < MA_{t-50,10}$
...	
6	$MA_{t-9,10} < MA_{t,10}$
7	$MA_{t-59,5} < MA_{t-55,5}$
8	$MA_{t-54,5} < MA_{t-50,5}$
...	
17	$MA_{t-9,5} < MA_{t-5,5}$
18	$MA_{t-4,5} < MA_{t,5}$

(b) Group2.

Figure 13: Group setting.

For simplification, we just use two groups with one agent in each group, and we denote them by *Agent1* and *Agent2* respectively.

(2) Prediction definition of the next changing direction.

In this section, we explain how to transfer the changed price value into integer directions for learning data. The definition is identical for different groups.

First, we set the direction number as $2 * m$. Directions $0...m-1$ denote an ascending trend, the next price will be greater than the present price. Direction $m...2*m-1$ denote descending trend. Second, we preprocess the raw data to get the max and min changed value between any two neighboring data, as given by Eq.(4–1). Third, we get the changed value between the present data and its next data by Eq.(4–2), and normalize it as in Eq.(4–3), which must be satisfied together with the inequalities.(4–4). Finally, we get the changing direction from Eq.(4–5).

$$maxChange = \max(\ |x_{t+1} - x_t|\ ) \qquad for\ all\ x_t$$

$$minChange = \min(\ |x_{t+1} - x_t|\ ) \qquad for\ all\ x_t \tag{4-1}$$

$$realChange = x_{t+1} - x_t \tag{4-2}$$

$$nChange = \frac{|realChange| - minChange}{maxChange - minChange} \tag{4-3}$$

$$0 \leq nChange \leq 1 \tag{4-4}$$

$$direction = \begin{cases} 0 & realChange \geq 0 \quad \& \quad nChange \in [0, \alpha_0] \\ \ldots & \\ m-1 & realChange \geq 0 \quad \& \quad nChange \in (\alpha_{m-1}, 1] \\ & \\ m & realChange < 0 \quad \& \quad nChange \in [0, \alpha_0] \\ \ldots & \\ 2*m-1 & realChange < 0 \quad \& \quad nChange \in (\alpha_{m-1}, 1] \end{cases} \tag{4-5}$$

For different experiment setting, we can set the parameter m as 1, 2 or 4, etc. Then the total direction number is two, four, or eight respectively.

We remark that, in the raw data, the changed value is not equally distributed from *minChange* to *maxChange*, but instead clustering near the *minChange*, with fewer data adjacent to the *maxChange*. To maintain a balance of data across every direction, we use a stepwise function. For different data sources, different values of the variable α in Eq.(4–8) are defined.

4.2.2 Hierarchical Learning Strategy

In this approach, all agents are identical XCSs, and are combined with a Q-Learning model. For each classifier in XCS, we make use of a *Q-Value* parameter in addition to the original parameters described in Chapter 2. The Q-Value parameter acts like a "guard" that allows the XCS to be activated. It is used to judge to what extent the agent can devote to recognize the present change pattern correctly. The parameter of XCS denotes to predict the changing direction.

That is to say, the multiple agents first try to identify the current trend of the market, and then apply locally-sound strategies to decide what action to take. For each region, only one agent is entrusted to make prediction.

Based on this architecture, we construct a hierarchical learning strategy as depicted in Figs. 14. In the higher hierarchy, we use a Q-Learning algorithm to discern which agent is more cognitive to the current market's changing trend. An optimal agent is then assigned to pursue the next prediction. In fact, the learning procedure is not quite identical to Q-Learning, but instead employs the Widrow-Hoff delta rule, given by Eq.(4–6). In the lower hierarchy, the selected agent is assigned to forecast the next up-down trend based on XCS. For each pattern, only one agent is entrusted to be suitable.

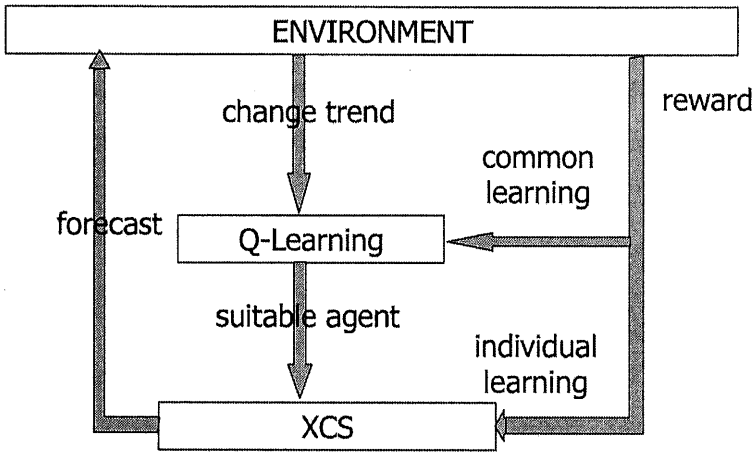


Figure 14: Framework of HXCS.

$$Q(s_{t+1}, a_t) = Q(s_t, a_t) + \beta * (r - Q(s_t, a_t)) \tag{4–6}$$

In details, for a given scene of the environment, each agent first perceives its own vision, and then constructs its MatchSet from the population. Next, the averaged Q-Value is calculated for each MatchSet. The agent with the maximum averaged Q-Value is considered to be the most accurate agent at recognizing the changing pattern. Thirdly, this assigned agent is then used to predict the next changing tendency using XCS. Finally, the resultant reward is used to update the *Q-Value* and the XCS parameters for this assigned agent.

As we have mentioned in Chapter 2, the environment recognized by XCS can be divided into single-step and multi-step problem. For these two kinds of settings, the XCS learning framework is also different. Here this forecasting application is defined as single-step problem.

After one forecasting step, it receives reward or zero immediately, and then the Learning Algorithm is pursued.

4.2.3 Improvement for XCS

In this proposal, since we use bit string to label stock changing waveforms, it is difficult for XCS to recognize all situations perfectly. It is not, however, necessary to match the changing trend pattern with each classifier perfectly in a prediction system. Thus when predicting the unknown data, we modify the matching rule from perfect matching to fuzzy matching, and set the mismatch probability to 10%. Suppose that the bit string is 18 bits in length, then the mismatch bit will be one bit or two bits. We then get the classifiers which match the perception bit string perfectly, and also those classifiers which have one or two bits mismatching. It may appear that some accuracy has been reduced. However, since we only inquire the trend, one or two bits differing does not greatly influence the overall forecast. At the same time, the fuzzy matching will lead to a prediction result derived from more related information.

4.3 Experiment1

In this experiment, we use the hit-rate to evaluate the prediction performance. Because in XCS we use integer values for changing direction, the prediction accuracy can be distinguished between perfect hit-rate and direction hit-rate. As shown in Eq.(4–5) (in Sect. 4.2.1), direction 0 means that the ascending range is smaller, while $m-1$ means a larger ascending range. When calculating the direction hit-rate, both direction 0 or $m-1$ mean ascending. The direction hit-rate means that, for the real changing direction0, all the forecasted directions from 0 to $m-1$ are deemed to be correct forecasting. The same situation occurs for directions m, $2*m-1$. In contrast, a perfect hit-rate means that the integer value of the forecasted changing direction has to precisely equal the real changing direction. Unless otherwise stated, in the following, prediction accuracy refers to the direction hit-rate.

Table 8: Statistic comparison of direction hit-rate among random prediction, trend-following strategy and HXCS (proposed method) (%).

	HXCS	trend-following	random-prediction
max	87.1	64	61
min	54.5	43	42
average	67.8	54.5	51.3

4.3.1 Comparison with Trend-Following Strategy

Although a lot of research has proved that stock market tendency is predictable [44, 45], there are still many people doubt on its feasibility, as reviewed in Sect. 4.1. To further verify the predictability, we firstly compare the forecasting performance of HXCS with that of trend-following strategy model and the random walk strategy. The raw data is TOPIX index from Mar.2000 to Jul.2004, obtained from Yahoo! finance. In this section, all the time-series data is daily closing price. We set the direction number of HXCS as two, then the direction is identified according to Eq.(4–7). Trend-following strategy model means that if the value of today is lower (higher) than yesterday's, then the tomorrow's price will be lower (higher) than that of today's. Random walk means that it does not use any reasonable strategy to forecast the next trend, up or down, but just sets it randomly. Figure 15 shows the direction hit-rate of average 100 days of these three models. To compare them clearly, we summarize the distinctive variable in Table 8. We find that the proposed HXCS outperforms trend-following strategy significantly, while trend-following strategy outperforms random walk slightly. We applied the same experiment on several other data, and obtained the similar result. From this point, we can conclude that forecasting the stock price is valuable to some extent, at least superior to the random walk. The deeper consideration strategy can obtain better performance. From now on, we will verify that our proposal has a superior level of performance by a series of experiments followed.

$$direction = \begin{cases} 0 & realChange \geq 0 \\ 1 & realChange < 0 \end{cases} \tag{4–7}$$

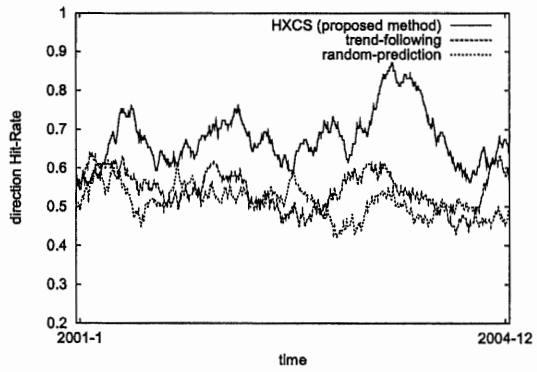


Figure 15: Trend comparison of direction hit-rate among random prediction, trend-following strategy and HXCS (proposed method).

4.3.2 Comparison with Single Agent

In this experiment, we set the prediction direction number as four; directions 0 and 1 are ascending, while direction 2 and 3 are descending, as given by Eq.(4–8). For a different data source, a different value of α may also be selected. Then the performance of HXCS is compared with one single agent(Agent1 and Agent2) separately. Each single agent learns using traditional XCS only.

We pursue our proposal on four indexes, NIKKEI, NASDAQ, TOPIX, HSI (HongKong Index), and also other eleven stocks, which are randomly selected from the components of KDDI. This data is taken from the daily closing price from Jan. 2000 to Dec. 2004, downloaded from Yahoo ! finance. The result is the average of 10 experiments with the same parameters settings. The population size is 100, learning period is 50 days, generation for one learning period is 10000, and total data to be forecasted is 1100. Interested readers can refer to Chapter 2 and [24] for the detailed parameter setting of XCS.

$$direction = \begin{cases} 0 & realChange \geq 0 \quad \& \quad nChange \in [0, \alpha] \\ 1 & realChange \geq 0 \quad \& \quad nChange \in (\alpha, 1] \\ & \\ 2 & realChange < 0 \quad \& \quad nChange \in [0, \alpha] \\ 3 & realChange < 0 \quad \& \quad nChange \in (\alpha, 1] \end{cases} \tag{4–8}$$

The average direction hit-rate is listed in Table 9. The columns of "Agent1" and "Agent2" represent the direction hit-rate using Agent1 and Agent2 separately. These are traditional methods using just a single agent, learning by the XCS algorithm. The column "HXCS"

Table 9: Comparison of direction hit-rate between single agent and HXCS on 15 financial data (%).

	Agent1	Agent2	HXCS
NASDAQ	66.9	70.8	73.8
HSI	68.1	69.7	72.9
TOPIX	63.9	67.2	69.6
NIKKEI	66.4	69.0	72.6
KDDI	64.1	70.0	72.2
HONDA	61.9	64.8	68.0
NEC	70.1	70.7	76.0
TOSHIBA	63.5	66.9	70.1
SONY	64.6	68.8	72.4
TOYOTA	61.7	66.3	68.9
Fuji Jyukou	65.3	68.7	71.3
Shin Nihon Setetsu	67.1	68.5	71.7
Nihon Express	65.0	68.2	70.3
Sanyo Denki	65.9	68.4	71.9
Shin Nihon Oil	65.9	67.5	71.8

gives the result of the proposed HXCS. From these results, we found that the proposal got around a 2-3% higher accuracy than that of a single agent.

We now consider why the Hierarchical XCS outperforms a single agent. By analyzing the final experiment result of the previous 10 experiments on TOPIX index, 1100 data for all, we summarized the prediction detail in Table 10. 'O' means that the agent forecasts successfully, while '×' means failure. For example, the results of row 'B' mean that, when forecasting a given changing trend, Agent1 fails, Agent2 succeeds, while the proposed HXCS also fails. Among the total 1100 data, 75 data has been forecasted in this situation.

In the upper four rows, HXCS selects Agent1 as its optimal agent, while for the lower four rows, HXCS obeys the strategy of Agent2. For example, in the first row labeled 'A', none of the three strategies could forecast successfully, with this situation occurring 67 times out of the total 1100 data. The same situation also occurs at row 'E'. Thus overall, out of the 1100 data to be forecasted, 168 data could not be forecasted correctly by any of the strategies. By contrast, in row 'D' and 'H', all of the strategies are successful.

Here we mainly focus on the rows labeled 'B' and 'C'. In row 'B', Agent1 fails, while Agent2 is successful. Because HXCS declares Agent1 more accurate at this position, the final result

Table 10: Comparison of direction hit amount between single agent and HXCS.

	Agent1	Agent2	HXCS	count
A	×	×	×	67
B	×	O	×	*75*
C	O	×	O	*103*
D	O	O	O	156
E	×	×	×	101
F	O	×	×	*110*
G	×	O	O	*165*
H	O	O	O	323

of HXCS will be consistent with Agent1. This results in a failure in HXCS's forecasting on 75 data. It seems that HXCS has failed at this point, but judgment should not be passed too early. In row 'C', the opposite situation occurs. Because HXCS obeys the policy of Agent1, it succeeds on 103 data. We can thus say that HXCS outperforms the single agent at this point. Although HXCS leads to some mistaken forecasts, it recognized more data successfully. Similar result are obtained by analyzing the lower four rows, where HXCS is obeying the policy of Agent2.

This is consistent with our objective. Given a changing direction trend, the Hierarchical XCS tries to determine which single agent outperforms the other. An appropriate suitable agent will be determined for each different trend.

Based on the same experimental result used above, we consider how the classifiers in Agent1 and Agent2 work. Table 11 lists the matched classifiers for a given scene of the two single agents. For each MatchSet, we first calculate the average Q-Value, and assign the agent corresponding to the larger value to be the selected agent for HXCS. Here, Agent1 is selected of course. Then for MatchSet of Agent1, the average preReward for each direction is calculated based on Eq.(4–9), weighted by fitness and given in the last column. The direction with maximum preReward is selected as the best forecasting direction. Thus direction 3, with an average preReward 1000, is selected as the optimal direction for this trend. In order to analyze the effectiveness of HXCS, we also calculate the optimal direction as determined by Agent2. With the same procedure, direction 1 is determined as the optimal prediction. Direction 3 means that the next price will decrease greatly, while direction 1 means it will increase greatly. In fact, from the raw data, we found that the real changing trend for the

Table 11: Classifier analysis in multiple agents.

condition	D	preReward	fitness	Q-Value	aPreReward
Agent1					
0*****0**0*******	0	185.5319	0.730582	1000	
*0*0*0***0*******	0	319.2584	0.635434	1000	247.7379
0**1*1*0****1**11*	1	594.3615	0.545388	1000	594.3615
0******00**110***1	2	582.5076	0.540335	1000	
0**1***0100*1*0***	2	639.7026	0.542199	1000	611.1543
0**01*00**0*****1	3	1000	0.307262	1000	
0*1010000*11**111	3	1000	1	1000	
0**01*00000*****1	3	1000	0.307262	1000	1000

condition	D	preReward	fitness	Q-Value	aPreReward
Agent2					
0000**00*000*00**0**0***	0	226.9555	0.942317	491.2768	
0***0*****0**1*0*****000	0	208.4168	0.941933	664.913	217.688
*0****0*0***00**00*****	1	746.6454	0.612831	794.545	
0*0*0*00**0*0*********0	1	213.2756	0.830154	915.4028	439.796
*0***0*0**00*****0**00*	2	453.911	0.927896	362.6214	
000000*0***0*****00***0	2	226.0137	0.939149	890.559	339.2755
0***000*0**0**0**000**01	3	446.1425	0.924962	8	
1*0**0000*0***0**0*0*0*0	3	437.5962	0.927377	8	
0**000***000**0**00****1	3	109.7412	0.928018	984.3156	331.0091

next day is a large descent. This result supports that we have obtained correct forecasting result based on Agent1.

$$avePreReward = \frac{\sum (preReward * fitness)}{\sum fitness} \tag{4–9}$$

4.4 Experiment2

4.4.1 Variable Reward Strategy

Generally, we use a profit sharing plan as the credit assignment [46], where a constant reward is paid to each classifier. However, as mentioned in Sect. 4.3.2, if the total direction number is set as four, directions 0 and 1 are appointed to the same changing tendency, but corresponding to a different changing range. If the real changing direction is 1, then a

forecasted direction 1 is better than 0, and should have received a higher reward. With this in mind, we induced a variable reward strategy for different prediction accuracies where the maximum reward is constant. The real reward for a prediction is in inverse proportion to the distance between the predicted and real direction. The more accurate a prediction is, the more effective it will be, and it survive with a high probability.

To analyze this, we apply our method on the NIKKEI, with a total of eight directions. Directions $0, 1, 2, 3$ represent an upward prediction for the next value, while directions $4, 5, 6, 7$ represent a downward. For the learning data, each direction value is derived on the basis of Eq.(4–10), which is extended from Eq.(4–5). The reward policy is based on Eq.(4–11). When the maximum reward is set as 1000, a prediction will receive a reward of 500 if a distance of 1 unit away from the real direction; and will receive a reward of 250 if the distance is 2 units, and so on. The experimental parameters are similar to Experiment1, except that the population size is changed to 250, and only one experiment has been performed.

$$direction = \begin{cases} 0 & realChange \geq 0 \quad \& \quad nChange \in [0, 0.133] \\ 1 & realChange \geq 0 \quad \& \quad nChange \in (0.133, 0.267] \\ 2 & realChange \geq 0 \quad \& \quad nChange \in (0.267, 0.534] \\ 3 & realChange \geq 0 \quad \& \quad nChange \in (0.534, 1] \\ & \\ 4 & realChange < 0 \quad \& \quad nChange \in [0, 0.133] \\ 5 & realChange < 0 \quad \& \quad nChange \in (0.133, 0.267] \\ 6 & realChange < 0 \quad \& \quad nChange \in (0.267, 0.534] \\ 7 & realChange < 0 \quad \& \quad nChange \in (0.534, 1] \end{cases} \tag{4–10}$$

$$r = \frac{reward}{pow\Big(2, abs(predictionDirection - realDirection)\Big)} \tag{4–11}$$

Table 12 displays the effectiveness of variable reward on NIKKEI. The rows describe the real changing direction, while the columns describe the predicted changing direction. Each element is the number of direction hit data from the total 1100 data. It may be observed that the data on the diagonal is slightly larger than that in other positions. The closer the data is to the diagonal, the larger it is. This is consistent with our reward strategy, for which the more accurate the prediction, the larger the reward can be obtained, and the larger the number of data that can be successfully forecasted. The last column is the hit-rate for each

Table 12: Direction hit detail by HXCS on NIKKEI.

	0	1	2	3	4	5	6	7	Hit-Rate (%)
0	17	27	21	9	29				71.8
1	23	46	32	14	32				78.2
2	18	22	41	43	40				75.6
3	12	6	22	72	28				80
4	48				21	21	19	7	58.6
5	30				21	22	23	9	71.4
6	27				21	27	53	41	84
7	17				7	15	20	97	89.1

real direction.

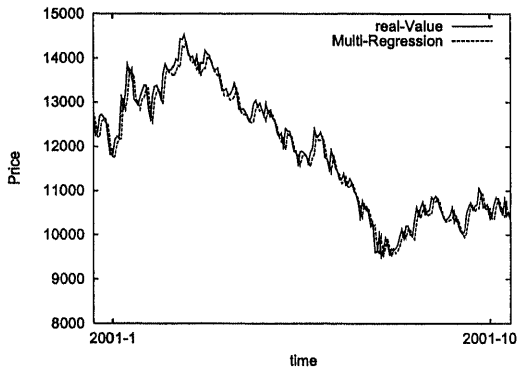
4.4.2 Comparison between Multiple Regression Analysis and HXCS

Having included a variable reward in Sec. 4.4.1, we plan to forecast the changing value. Although our proposal in this thesis mainly focuses on changing direction prediction, reverting the direction prediction to changing value prediction will help us illustrate its efficiency more clearly. Based on the inversion processing of Eq.(4–10), we can revert integer changing direction to changing value.

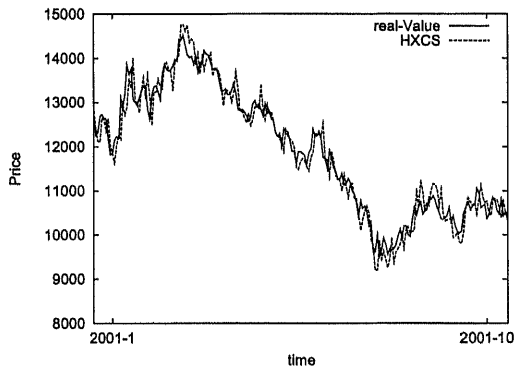
We compare the simulation of HXCS with that of a single agent, and also with that of Multiple Regression Analysis (MRA) [33], a traditional statistic algorithm. The raw data is the NIKKEI index from Jan. 2001 to Oct. 2001, with total 200 data. In MRA, multiple variables like opening, closing, high, and low price have been involved. We use Excel 2003 to calculate the estimators of coefficients for each parameter, which are given by Eq.(4–12). The simulation curve has been shown in Figs. 16, compared with real value respectively. The statistics comparison is listed in Table 13 as well.

$$\begin{aligned} prediction = & -0.06074 * open + 0.351238 * high \\ & - 0.01521 * low + 0.707544 * close + 150.68 \end{aligned} \tag{4–12}$$

From Figs. 16, we found that, the Multiple Regression Analysis outperforms HXCS with less deviation to real value, and the coherent result appears in Table 13. Compared with HXCS, the MRA has less MAE (Mean Absolute Error), RMSE (Root Mean Square Error) and MRE (Mean Relative Error) compared with HXCS(defined as Eq.(4–13)). However, the



(a) Multiple Regression Analysis.



(b) HXCS.

Figure 16: Simulations of Multiple Regression Analysis and HXCS (proposed method).

Table 13: Statistics comparison between Multiple Regression Analysis and HXCS.

	MRA	HXCS
MAE	182	242
RMSE	239	315
MRE	0.015	0.020
Hit-Rate (%)	54	72

opposite result appears for the hit-rate in agreement with expectation. We recall that our objective is to get a higher direction hit-rate, rather than perfect simulation accuracy. By analyzing the prediction data in detail, we obtained the following conclusion. For a real upward trend with changed value a, we forecast an upward with changed value $3 * a$ by HXCS, and a downward trend with changed value $0.5 * a$ by MRA. The simulation deviation of HXCS is then $2 * a$, while that of MRA is $1.5 * a$. Since here we placed emphasis on the correct forecasting of tendencies, on this basis we confirm that the HXCS is superior to MRA. Although the deviation of MRA is smaller, it is minor. The results show that HXCS gets a higher direction hit-rate, but also higher deviation than MRA.

$$MAE = \frac{1}{N} * \sum_{i=1}^{N} |x_i - \bar{x}|$$

$$RMSE = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

$$MRE = \frac{1}{N} * \sum_{i=1}^{N} \frac{|x_i - \bar{x}|}{x_i} \tag{4-13}$$

4.5 Discussion

These experimental results verify the effectiveness of HXCS in forecasting the next fluctuation trend. It is superior to not only traditional trend-following strategy, but also a single agent model. We now discuss some aspects of the advantage and limitations of the proposed HXCS.

(1) Set suitable learning strategy for an agent.

We use two heterogeneous agents within a hierarchical learning structure, in order to overcome the shortcoming of one single agent arising from its narrow vision. However, there are some issues which should be discussed regarding the present setting. First, there are several variables associated with a real stock market, such as closing, high, low and opening price, volume, and so on. All these factors affect the whole market. However, here we only consider daily closing data, which does not completely represent real fluctuations. If the agent vision expands to other variables, the performance may improve.

Secondly, the perception of present two agents is defined experimentally. We could not confirm their effectiveness theoretically.

In conclusion, the two problems above focus on how to define an agent's perception effectively, with fewer agents obtaining as much valuable information as possible. This is the first area of investigation for future work.

(2) Acquire optimal agent among multiple agents.

In this proposal, we use the Q-Learning method for the upper hierarchy to select the more appropriate agent from two candidates. If we use more than two agents, how to combine these agents in order to absorb useful knowledge is another issue which should be considered.

(3) Predict changing tendencies with one single agent.

Based on the foundation of the upper hierarchy implemented with Q-Learning, the optimal agent can give a more accurate prediction of fluctuating changing trends by means of its local model employing XCS. This is the highlight of HXCS.

(4) Forecast changing value with one single agent .

Besides the changing direction prediction, variable reward strategy helps provide an approximate prediction value. However, HXCS is inferior to MRA when judged on the basis of statistical analysis. This is because HXCS is orientated toward direction forecasting. Good direction forecasting may be associated with a larger error in the changing value. As an improvement, we can further refine the direction forecasting. According to our experience, the value prediction will then obtain a higher accuracy. However, accompanying this, the learning complexity of XCS will significantly increase. Thus selecting an optimal compromise between accuracy and learning complexity is another issue which may be addressed.

4.6 Conclusions

In this proposal, we involve a Hierarchical XCS for time series forecasting problem. Different from general used approaches, in HXCS, we focus on the changing direction forecast, instead of the quantitative prediction. By experiments, we found that it is superior to not only traditional trend-following strategy, but also the same strategy with one single agent and one layer learning.

Based on the advantage and limitation, as we have discussed above, we will focus our attention on the cooperation of a higher number of agents and changing value prediction in the future. Finally, applying HXCS on time-series data from another area forms our next project. It is a wider field that can benefit mankind.

Chapter 5

Conclusion

Based on the related work, a series of Classifier System has been proved to be a satisfactory platform, as a reliable robust Machine Learning architecture, which combines RL, EC and other heuristics to produce adaptive systems. Its advantage has been approved greatly, especially for the complex problem. In this thesis, we mainly focused on two aspects originating from XCS. One is to benefit the structure of XCS by adopting adaptive internal space; the other is based on hierarchical invocation of classifiers. They correspond to multi-step and simple-step problem respectively.

(1) The first proposal centers on Non-Markov environment confronted with autonomous agent control.

We proposed an adaptive XCSM system (AXCSM) for the benchmark problem of Maze with Non-Markov property, which involved a hierarchical structure to adapt to the variable length of memory space, ranging from 0 bit to the maximum length. The experimental results show that the adaptive XCSM converges to the same optimal strategy as XCSM, but within shortened search space. Based on the analysis of experiment result, we found out that the corresponding register content changed as the agent's perception varied. As the learning proceeded, a sub-optimal adaptive memory length has been obtained. The future work will be continued on more complex problem, even for a real-value environment, in order to observe the changing mechanism of variable memory, and seek higher efficiency.

(2) The second proposal focuses on the stock market forecasting problem.

When analyzing a real stock market, in general, we pay more attention on qualitative changes first, and then on tracing the quantitative changes. In order to forecast the stock market, we propose a Hierarchical XCS system, HXCS. Each agent learns through a hierar-

chical structure, by applying RL approach to XCS. With the combination of multiple agents, the narrow vision of one single agent can be overcome in order to predict the next changing tendency, up or down. Through experiments on several well-known stock indices and stock markets, the multiple agents can cooperate with each other to acquire higher direction hit-rate than just using one single agent. We just used two agents in this proposal, based on one single kind of data in stock market(daily closing price). How to involve more variables from raw data, or more learning strategy of agents will be the prospect in the future.

Besides the tendency prediction, we introduced a variable reward strategy in different prediction accuracy, which helped providing a value prediction. However, the proposal is inferior to Multiple Regression Analysis on statistic analysis, since we pay attention on direction first. The right direction forecast may be followed by a great error in changed value. By increasing the direction numbers, the value prediction will become more efficient, but the learning complexity of XCS will increase as well. This is another deficiency in achieving higher performance.

In summary, for many tasks in Artificial Intelligence, problem solving is to get the regularities on expressing a given problem. As more difficult problems are considered, the size of search space grows quickly, and it becomes important to summarize regularities into compact knowledge base. The Classifier System, as a common used intelligent method with compact representation, will attract more attention in real world.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my thesis advisor, Professor Tomoharu Nagao, for his powerful guidance, support and attention from the beginning till end over the past three years. I am lucky to have a kindhearted teacher like him. Sometimes I got depressed because I could not devise ideas after long time studying, and nearly lost confidence. It was his encouragement that gave me strength and has kept me continuing the research. I can not forget his helps and encouragement. I would also like to thank the members of my thesis committee, Professor Hiroshi Arisawa, Professor Naoyoshi Tamura, Professor Tatsunori Mori and Professor Takashi Tomii for their helpful advice and comments.

I would like to send my special thanks to Monbukagakusyou, who granted my scholarship. This scholarship is a great support to my family and my tuition so that I could concentrate on my research without doing a part-time job. Without it, I think I could not graduate now.

I am also grateful to the present and former members of the Nagao Laboratory, for the useful discussions and constructive comments, and teaching me Japanese language.

I want to say thanks to those, some of whom I even cannot know their names, for their giving me helps.

Finally, I would like thank my husband and my parents, for their love and for giving me support and encouragement that has enabled me to finish the studies.

References

[1] W. Hsu and S. Gustafson, "Genetic Programming and Multi-Agent layered learning by reinforcements," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), New York, July 2002.

[2] W. Lee, J. Hallam, and H. Lund, "Applying Genetic Programming to evolve behavior primitives and arbitrators for mobile robots," Proceedings of the IEEE International Conference on Evolutionary Computation, 1997.

[3] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada, "How to evolve autonomous robots: different approaches in evolutionary robotics," Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, July 1994.

[4] K. Tazawa, T, Nagao,, "Behavior control of an autonomous agent using flexibly connected Neural Network; FCN," IPSJ (in Japanese), vol.45, pp.991–1000, 2004.

[5] H. Kataoka, A. Hara, T. Nagao, "Action control of an agent using adaptive GP-Automata," IPSJ, vol.44, pp.2390–2400, 2003.

[6] M. Wiering, J. Schmidhuber, "HQ-Learning," Adaptive Behavior, vol.6(2), pp.219–246, 1997.

[7] X.Y. Yu, S.Y. Liong, V. Babovic, "EC-SVM approach for real-time hydrologic forecasting," Journal of Hydroinformatics, vol.6(3), pp.209–223, 2004.

[8] S. Dablemont, G. Simon, A. Lendasse, A. Ruttiens, F. Blayo, M. Verleysen, "Time series forecasting with SOM and local non-linear models-Application to the DAX30 index prediction," WSOM'2003 Proceedings of Workshop on Self-Organizing Maps(Japan), pp.340-345, Sep.2003.

[9] H. White, "Economic prediction using Neural Networks: the case of IBM daily stock returns," Proceedings of the IEEE International Conference on Neural Networks, July 1988.

[10] P. Lanzi, "Extending the representation of classifier conditions. Part I: from binary to messy coding," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), pp.337–344, 1999.

[11] P. Lanzi, "Extending the representation of classifier conditions. Part II: From messy coding to s-expressions," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), pp.345–352, 1999.

[12] S. Liu and T. Nagao, "Adaptive XCSM for perceptual aliasing problems," Lipo Wang, Ke Chen, and Yew Soon Ong. eds., Proc of ICNC 2005-FSKD 2005, LNCS, pp.566–571, Springer-Verlag, 2005.

[13] A. Wada, K. Takadama, K. Shinohara, and O. Katai, "Analyzing real-valued learning classifier systems (in Japanese)," JSAI, vol.20, no.1, pp.57–66, 2005.

[14] Barry, Holmes, and Llora, Data mining using Learning Classifier Systems, ch. Applications of Learning Classifier Systems, Springer-Verlag, Berlin, 2004.

[15] G. Armano, A. Murru, and F. Roli, "Stock market prediction by a mixture of genetic-neural experts," International Journal of Pattern Recognition and Artificial Intelligence(IJPRAI), vol.16, no.5, pp.501–526, 2002.

[16] Y. Ikeda, J. Lv, and S. Tokinaga, Analysis of fractal and Chaos of stock market based on Multi-Agent, ch. The new field of mathematical finance and its application (in Japanese), Industry Publisher, Japan, 2004.

[17] J. Holland and J. Reitman, Cognitive systems based on adaptive algorithms, ch. in Waterman, D.A., Hayes-Roth, F. (eds.), Pattern-directed inference systems, Academic Press, New York, 1978.

[18] S. Smith, A Learning System based on Genetic Algorithms, Ph.D. thesis, University of Pittsburgh, 1980.

[19] J. Holland and J. Reitman, Cognitive systems based on adaptive algorithms, ch. In D.A. Waterman and F. Hayes-Roth, editors, Pattern-directed inference systems. New York: Academic Press, 1978. Reprinted in: Evolutionary computation: the fossil record, IEEE Press, New York, 1998.

[20] L. Booker, D. Goldberg, and J. Holland, "Classifier systems and Genetic Algorithms," Artificial intelligence, vol.40, pp.235–282, 1989.

[21] A. Barry, XCS performance and population structure within multiple-step environments, Ph.D. thesis, Queens University Belfast, 2000.

[22] S. Wilson, "Knowledge growth in an artificial animal," Grefenstette, J.J. (ed.) Proceedings of the First International Conference on Genetic Algorithms and their Applications(ICFGA85), Lawrence Erlbaum Associates, Pittsburg,, July, 1985.

[23] S. Wilson, "ZCS: A zeroth level Classifier System," Evolutionary Computation, vol.2, pp.1–18, 1994.

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

[25] M. Butz and S. Wilson, "An algorithmic descriptions of XCS," Advances in learning classifier systems, LNAI, Berlin, Berlin:Springer-Verlag, 2000.

[26] S. SaxonW and A. Barry, "XCS and the Monk's problems," Wu, A.S.(ed.), Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program, 1999.

[27] L. Bull, Applications of Learning Classifier System, Springer-Verlag, 2004.

[28] Lanzi, S.W. Wilson, Get Real! XCS with continuous-valued inputs, ch. P.L., Stolzmann, W., Wilson, S.W.(eds), Learning Classifier Systems: Introduction to Contemporary Research, vol. 1813 of LNAI, 209-220, Springer-Verlag, Berlin, 2000.

[29] D. Cliff and S. Ross, "Adding temporary memory to ZCS," Adaptive Behavior, vol.3, no.2, pp.101–150, 1995.

[30] P. Lanzi and S. Wilson, "Toward optimal Classifier System performance in Non-Markov environments," Evolutionary Computation, vol.8, no.4, pp.393–418, 2000.

[31] P.A. Crook and G. Hayes, "Learning in a state of confusion: perceptual aliasing in grid word navigation," Proceedings of Towards Intelligent Mobil Robots(TIMR 2003)4th British Conference on(Mobile) Robotics, 2003.

[32] B. Malkei, A random walk down wall street: including a life-style guide to personal investing, W.W. Norton Company, New York, 1999.

[33] G. Maddala, Introduction to econometrics, Macmillan Publishing Company, New York, 1998.

[34] F. Castiglione, "Forecasting price increments using an Artificial Neural Network," Advances in complex systems, vol.4, pp.45–56, 2001.

[35] C. Kuan and T. Liu, "Forecasting exchange-rates using Feedforward and Recurrent Neural Networks," Journal of Applied Econometrics, vol.10, pp.347–364, 1995.

[36] F. Allen and R. Karjalainen, "Using Genetic Algorithms to find technical trading rules," Journal of financial economics, vol.51, pp.245–271, 1999.

[37] S. Mahfoud and G. Mani, "Financial forecasting using Genetic Algorithms," Applied artificial intelligence, vol.10, pp.543–565, 1996.

[38] A. Abraham, B. Nath, and P. Mahanti, "Hybrid intelligent systems for stock market analysis," The 2001 International Conf. on Computational Science (ICCS 2001), LNCS 2074, San Francisco, USA, pp.337–345, 2001.

[39] Y. Ikeda and S. Tokinaga, "Chaoticity and fractality analysis of an artificial stock market generated by the Multi-Agent systems based on the Co-Evolutionary Genetic Programming," IEICE, vol.E87-A, no.9, pp.2387–2394, 2004.

[40] S. Ogino, A. Hara, and T. Nagao, "An analysis and construction considering chaotic components of Multi-Agent based artificial stock markets (in Japanese)," IPSJ, vol.46, no.6, pp.1516–1526, 2005.

[41] S. Schulenburg and P. Ross, An adaptive agent based economic model, ch. Learning Classifier Systems: From Foundations to Applications, pp.265–284, LNAI 1831, 2000.

[42] S. Schulenburg and P. Ross, "Explorations in LCS models of stock trading," In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, Advances in learning classifier systems, Vol.2321 LNAI, Berlin, pp.150–179, Springer-Verlag, 2002.

[43] S. Schulenburg and P. Ross, "Strength and money: an LCS approach to increasing returns," In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, Advances in learning classifier systems, Vol.1996 LNAI, Berlin, Springer-Verlag, 2001.

[44] W. Brock, J. Lakonishok, and B. Lebaron, "Simple technical trading rules and the stochastic properties of stock return," Journal of finance, vol.47, pp.1731–1764, 1992.

[45] N. Gershenfeld and A. Weigend, The future of time series: learning and understanding, pp.1–70, Addison Wesley Publishing Company, 1993.

[46] J.J. Grefenstette, "Credit assignment in rule discovery systems based on Genetic Algorithms," Machine Learning, vol.3, pp.225–245, 1988.

Published Papers

Full Paper:

Shumei Liu, Tomoharu Nagao, "Hierarchical XCS and Its Application to Financial Time Series Forecasting,", IEEJ Transactions on Electrical and Electronic Engineering (TEEE), to appear in Nov. 2006 (accepted in Aug. 2006)

International Conference:

Shumei Liu, Tomoharu Nagao "An Adaptive XCSM for Aliasing Problems," ICNC ' 2005-FSKD ' 2005, Vol.3612 LNCS, pp.566-571, Springer-Verlag, 2005

Domestic Conferences:

Shumei Liu, Tomoharu Nagao, "Adaptive XCSM for Perceptual Aliasing Problems", The 18th Annual Conference of the Japanese Society for Artificial Intelligence, 2004