Heuristic-based Genetic Operation in Classifier Systems

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

(Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

NUGROHO FREDIVIANUS, M.Sc.

Tag der mündlichen Prüfung:	27. Februar 2015
Referent:	Prof. Dr. Hartmut Schmeck
Korreferent:	Prof. Dr. Rudi Studer

2015 Karlsruhe

ABSTRACT

This thesis focuses on improving the Accuracy-based Learning Classifier System (XCS), an online Machine Learning technique that keeps its knowledge in a population of rules. Introduced in 1995, the system has been developed to a number of variants. However, most of them still face an issue of suffering from a slow learning rate. Several investigations have addressed this matter, yet with no convincing results. Therefore, this thesis introduces a novel technique named Rule Combining (RC), with an ability to provide a better solution. The new approach specifically replaces the operators of the Darwinian Genetic Algorithm (GA) in producing a number of better children from some selected parents.

Different from the crossover and mutation in GA, RC does not employ any randomization processes in creating offspring. Instead, it adopts the inductive reasoning (induction) concept along with some other notions to build the children. So, the system considers only the knowledge that is already available, and draws some conclusions out of it. At this point, one can say that the quality of RC's results highly depends on the current rules. A comprehensive knowledge in XCS's population would lead to a number of valid conclusions.

Improving the learning rate is not the only advantage for XCS. RC also assists the system to increase its efficiency, denoted by the significant reduction of space that is required to maintain the knowledge. XCS with RC (called XCS-RC) shows a better learning capability while requiring a smaller size of population. Moreover, the proposed system is capable of solving various tasks that the original XCS (with GA) tackles unconvincingly, both in handling inputs in binaries and real values.

A further improvement of XCS is achieved by an additional capability of performing a feature selection operation. This ability allows the system to evaluate the received input and to conclude for each part of the information the relevance level with respect to the learning process. The parts with no relevance are filtered out, letting the system run more efficiently with respect to speed and space requirements. Such operation can only be performed by XCS having a set of reliably general and accurate rules. Therefore, adding feature selection to the original XCS would be considered as a huge risk, since the system potentially removes relevant information.

Finally, this thesis provides a set of comparative evaluations based on a series of experiments using typical test problems. The results show that XCS-RC is significantly more superior compared to the original XCS in terms of learning speed and space efficiency.

ACKNOWLEDGEMENTS

One funny thing I felt while writing this thesis is an urge for traveling through time towards the past, long way back to the ancient Greek era. Physically, such thing is still impossible. However, performing it mentally was necessary in order to familiarize the ideas that I wanted to explain. Diving into the haystack of moments where humans built the foundation of science was a priceless experience. However, one might not see that as a good idea for writing the beginning of a doctoral thesis. And I knew that I should be alarmed when my doctoral supervisor also felt that way the first time I discussed my idea with him.

So, firstly, I would like to express my gratitude to Prof. Dr. Hartmut Schmeck for all of his supports while I was walking in the path of scientific works. There were moments where I can barely move forward and he showed some patience. And by the time I was prepared to run towards the finish line, he also provided necessary suggestions that were very helpful to avoid any accidents. And it is one of the best moments in my life when he finally agreed my idea of placing the history of science as a foundation of the thesis.

I would also like to thank the second reviewer of my thesis, Prof. Dr. Rudi Studer, for being an important role on the examination committee. Furthermore, many thanks to Prof. Dr. Frank Schultmann and Prof. Dr. Wolf Fichtner for their participations as the examiner and the chairman on the examination committee, respectively. I suppose that we had a nice discussion in the forum, and I am grateful for that.

Starting to write a scientific work from scratch was not easy. Having supportive colleagues at the Efficient Algorithms research group was a big help indeed. Therefore, I would like to thank my predecessor in the OCCS project Dr. Urban Richter, for his time and willingness to discuss many things that were required. Also, Dr. Holger Prothmann that was actively involved in several deep discussions in the early phase of my thesis. For their assistance that kept me survive at this business a few years ago, I would like to thank Prof. Dr. Sanaz Mostaghim, Micaela Wünsche and Christian Hirsch. Finally, many thanks to Dr. Lukas König for his life-saving suggestions near the end of the process.

From the Learning Classifier Systems community, I would like to thank Dr. Stewart Wilson for his endorsement on my investigations. We had a nice discussion at the GECCO 2012 conference in Philadelphia. You gave me a number of suggestions regarding the way I explain my contribution scientifically, and your argumentations lead me to pick a proper title for this thesis.

Moving outside the scientific field, I would like to thank my family both in Indonesia and Germany. Although I could not hear your voice all the time, I was, and still am, sure that you have supported me in several ways one might not be able to imagine. My mother Sri and my father Suparno, my wife's parents Herniti and Tarsobidi, my brother Ary and my sister Offi. But, if anyone was capable of making me feel incomplete, it is my wife Mira. Thanks for all your support and love, both in easy and difficult times, including when you had to fully take care of our children Aida, Raihan and Firdaus, which I am also grateful for them. Last but not least, Alhamdulillah, all praise to Him, for letting me achieve all these wonderful things, also for His guidance and tests that make me a better and stronger person from time to time.

Karlsruhe, April 2015

Nugroho Fredivianus

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

$[A]$	-	Action set
$[C]$	-	Combining set
$[M]$	-	Match set
$[P]$	-	Population of knowledge
$[P]_{opt}$	-	The optimal $[P]$
ANN	-	Artificial Neural Network
CB	-	Checkerboard task
cl	-	A classifier
cl_*	-	A classifier candidate
GA	-	Genetic Algorithm
LCS	-	Learning Classifier System
MP6	-	Multiplexer task with six elements of inputs
MP11	-	Multiplexer task with 11 elements of inputs
MP20	-	Multiplexer task with 20 elements of inputs
NASA	-	National Aeronautics and Space Administration
OC	-	Organic Computing
RC	-	Rule Combining
SuOC	-	System under Observation and Control
XCS	-	Accuracy-based Learning Classifier System, as in [35, 36]
XCSR	-	XCS with real inputs, as in [37]
XCSS	-	XCS with Specify, as in [20]
XCS-RC	-	XCS with Rule Combining, as in [12]

CHAPTER 1 INTRODUCTION

"The only source of knowledge is experience."

(Albert Einstein)

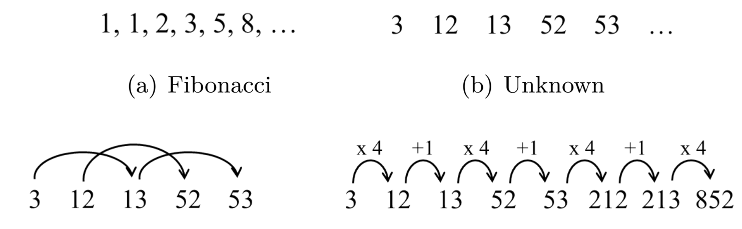
By the time Einstein made the statement, he surely did not talk about machines. His sentence was solely focused on human, despite the possibility that he might have predicted that one day machines will be capable of maintaining their very own knowledge. Tens of years afterwards, those words became applicable to various machine learning techniques. Experience becomes a vital factor in designing an intelligent machine's knowledge, either through its own learning process or initially planted by human.

Hundreds of years before Einstein, Aristotle made a similar statement within that matter. He introduced a paradigm which is later called *inductive reasoning* (or *induction*) by stating that knowledge can only come from the experience. At his time, the common religion was polytheism and people strongly held the belief that Athena, the goddess of knowledge, had a full authority of distributing her power to the human.

The concept stated by Aristotle implies the idea that every human observes and makes conclusions by discovering any underlying patterns within the collected experience. For example, a small child may come to a conclusion by walking to the kindergarten every weekday morning with one of his parents. Every time they are about to cross a street, the parent always asks him to stop when the light is currently red, waiting until the sign becomes green. After experiencing such occasion quite regularly, he would learn that a red light means that they should not go crossing the street, while a green stating otherwise.

In mathematics, induction provides a solution for coping with various assignments. For instance, Figure 1.1(a) presents a typical *Fibonacci* sequence starting with two 1s and followed by a series of numbers. Each following number is determined by calculating the sum of its two predecessors. By only observing some of the earliest values, one can theoretically discover the whole sequence.

The second sequence depicted in Figure 1.1(b) shows an unknown set with five initial numbers, i. e., 3, 12, 13, 52, and 53. After observing these values, one has to find the specific pattern to figure out the following members. A possibility is given by assuming that the sequence is composed using an alternating relation where the numbers 3, 13 and 53 come from one group and the rest belong to the other (see Figure 1.1(c)). However, as depicted in Figure 1.1(d), the most probable pattern here is represented by two alternating



(a) Fibonacci (b) Unknown

(c) Possible connection from the unknown (d) One of the valid solutions

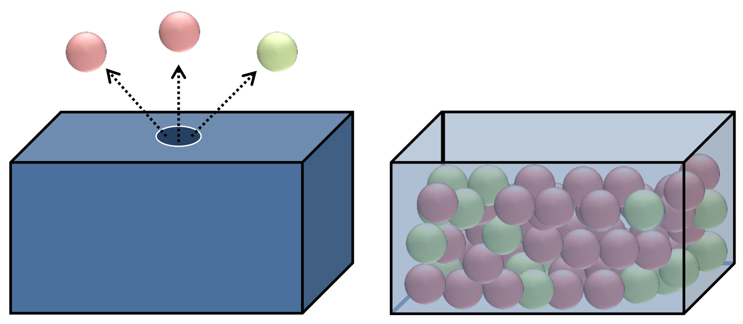
Figure 1.1: Sequence

calculations. After the initial value "3", the direct successor is determined by multiplying it by four, making it "12". Then, the number "13" comes as a result of an increment "12 + 1", and those operations are executed one after the other. After discovering the pattern depicted in Equation 1.1, calculating all unknown values is possible.

$$n_i = \begin{cases} 3 & \text{if } i = 1, \\ 4 \times n_{i-1} & \text{if } i \text{ is even} \\ n_{i-1} + 1 & \text{if } i \text{ is odd, } i > 1. \end{cases} \tag{1.1}$$

Both examples suggest that a small number of samples is sufficient to discover a pattern in a reasonable way. In the Fibonacci case, performing an induction on the given sequence leads to a mathematical formula $n_{i+2} = n_i + n_{i+1}$ after two initial 1s. Then, the conclusion is generalized by assuming that the whole sequence can be represented by the equation. Such technique is valid for determining the unknown values of the second sequence as well, using its respective formula.

Another illustration is provided by the so-called *Black Box* scenario, involving a group of physical matters as depicted in Figure 1.2. An observer is given three opportunities to take some unknown contents from a black box, one individual at a time. Shortly afterwards, he discovers three ball-shaped items in two different colors. One of them is green while the others are red (see Figure 1.2(a)).



(a) Three samples of the population (b) Conclusion

Figure 1.2: Concluding the content of a "black box" (an unknown population)

At this point, a conclusion can be made based on the current samples. It is logical that the observer tends to conclude that the box is filled with balls. Furthermore, he may also believe that the red individuals have a greater share than the green as illustrated by Figure1.2(b), with a comparison of somewhere near 2:1. Here, the three samples play the role as a set of representatives of the whole population, and a conclusion is made based upon them.

This thesis proposes a new method, adopted from that induction concept. The paradigm leads to the composition of a novel learning technique based on a similar line of thoughts. As humans collect their knowledge from time to time by making conclusions out of their experience, machines can be designed to behave similarly with some limitations. This introductory chapter describes an overview regarding the process of building the scientific knowledge, and how it can be implemented to the machine learning, especially those operating as classifier systems.

The remainder of this chapter is composed as follows. Firstly, a brief history of inductive reasoning is presented. Then, a short description of the background is given in Section 1.2, followed by an explanation regarding the motivation. Section 1.4 highlights the contribution of this thesis while Section 1.5 provides a guide for readers. Finally, an acknowledgment ends this chapter as a credit to those who gave meaningful contributions in the composition of this thesis.

1.1 History

The concept of induction began long way back to the ancient Greece era. Plato (428/427 - 348/347 BC) and one of his students, namely Aristotle (384 - 322 BC), had a long debate over the philosophy of science [22]. Plato, strongly affected by the current Greek belief and wisdom, stated that each human is born with a set of hidden knowledge in his memory. Then, as time goes by, his life experience gradually uncovers the knowledge. However, the brilliant pupil had another thought stating the opposite. Aristotle said that every human does not know anything at the time he was born. Afterwards, each life event presents a chance for observing, gathering information, and then concluding the result in a so-called *knowledge* [23].

Since then, *philosophy* and *science* established their places in the civilizations at two different stand-points. Aristotle was acknowledged as the father of science, while the father of philosophy was entitled to Plato. Like a pair of twins, philosophers play their roles by producing many thoughts, inspiring the scientists to investigate the unknowns and to always pursue the logical explanations. At this point, Aristotle had brilliantly founded the systematical thinking used by scientists until today, called *inductive reasoning* or *induction.*

Centuries afterwards, in the Islamic golden age, a scholar namely Ibn al-Haytham or Alhazen (965 - 1040 AD) made a breakthrough in the field of optics [29]. He was a pioneer in studies on light refraction, which later gives a way to the invention of a lens. However, his biggest contribution to science was realizing in his work the first scientific implementation of Aristotle's induction paradigm. With that, Alhazen established a way in conducting investigations which is now known as the first practical scientific method: observation, hypothesis, followed by experiments.

Such method led Galileo Galilei (1564 - 1642 AD) to one of the most notable discoveries in science [34]. His work in astronomy has indeed given the strong foundation for induction. A series of observations encouraged him to publish the theory of heliocentrism, stating that the earth revolves around the sun. The conclusion was based on the fact that planet Venus actually has light and dark phases, which would never happen if geocentrism was correct. This idea confirmed the statements previously made by at least two notable astronomers,

i. e., Ibn al-Shatir (1304 - 1375 AD) [26] and Nicolaus Copernicus (1473 - 1543 AD) [19]. However, Galileo's idea got harsh critics from other scholars as well as religious groups. At that time, geocentrism was considered as the scientific truth and also the common belief, stating that the earth is the center of the universe.

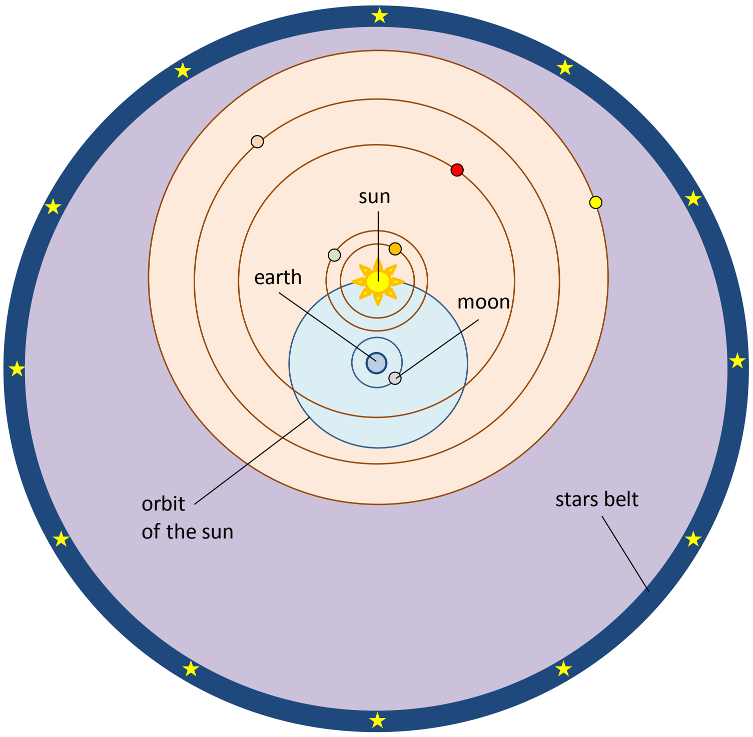


Figure 1.3: An illustration of the Tychonic system, as proposed by Tycho Brahe

Slightly before Galileo, another universe theory was introduced by Tycho Brahe (1546 - 1601 AD), known as the *Tychonic System* [31]. As depicted in Figure 1.3, the system could be addressed as a combination of geocentrism and heliocentrism. According to Brahe, the sun and the moon revolve around the earth while each of the other planets plays a role as a satellite of the sun. Brahe came to this theory after several series of observations. His work was considered having the most accurate calculations with no involvement of telescopes. Like Galileo, Brahe came to such conclusion by practicing inductive reasoning. Despite the possibility that he was being accommodative to the opinion held by the religious and political authorities at the moment, his findings followed the inductive reasoning paradigm.

Not only in astronomy, Galileo also conducted an investigation in the field of physical science using Alhazen's scientific method. In the 17*th* century, he published a theory called *Galilean Equivalence Principle*. It stated that any physical matter originally has the same velocity when falling to the ground. Galileo argued that the differences were actually caused by the atmosphere instead of the falling objects themselves. The conclusion came after investigating the fall of a set of balls made from different materials. His experiments showed that all balls reached the ground at considerably the same time, after being dropped from the leaning tower in Pisa, Italy.

Galileo's theory was further developed by Isaac Newton (1643 - 1727 AD). He agreed to Galileo's thought and pushed even further to establish the *Theory of Gravity*. Centuries afterwards, this line of thought was backed up even more convincingly in 1971 by a NASA astronaut Dave Scott who landed at the moon. The experiment was done in that non-atmospheric field using a hammer and a feather, where both fell off his hands and touched the sands at approximately the same time.

All the aforementioned investigations by Alhazen, Brahe, Galileo and Newton provide some historical milestones stating that knowledge is created inductively after finding one or more patterns throughout several instances. Using the same paradigm, various kinds of balls, fruits (like apples as in Newton's famous story regarding the theory of gravity), a hammer and a feather behaved identically. The vertical positions of those objects are affected solely by gravity. With only a small number of examples, conclusions are made and the result could be generalized to nearly all physical entities.

Unlike the ancient Greece era nor the years when Galileo conducted his experiments, nowadays, scientific observations can be done using machines, namely computers. As the technology gets more sophisticated, the demand of accuracy as well as safety become the main issues. Maintaining millions of data, investigating some dangerous and radiated areas, or simply measuring the speed of a microscopic particle, are some instances of modern activities assisted by machines. Yet a question follows, is it possible for computers to do something more analytical, such as discovering a pattern out of the results of observation?

In 1950, a well-known research work namely *Turing Test* was published [32]. It investigated the possibility of designing a machine that is capable of imitating human's behavior. The experiments were conducted by asking a human interrogator to identify whether a "person" in another room is a human or a machine. The interaction was done by exchanging texts to ensure that the interrogator's judgement was based on the content of the opponent's response. Several years afterwards, the issue brought by this "imitation game" is very much still alive, along with numerous arguments, both supporting and criticizing.

The topic presented in this thesis takes a notion from the idea of imitating human's behavior, particularly in the process of forming a set of knowledge. Implementing Plato's concept to design a mechanism that allows a machine to have pre-birth memory followed by a series of "internal discovery" would be very difficult. Hence, Aristotle's paradigm is considered more suitable for designing intelligent machines, i. e., equipping them with the capability of building their very own knowledge, inductively.

Designing machines to imitate humans in performing inductive reasoning is a way to build intelligent entities. However, the validity of an induction largely depends on the comprehensiveness of the observation. Practically, machines should be equipped with a capability of detecting invalid conclusions. Such process plays a very important role in performing induction, and therefore has to be handled properly. The history of science provides numerous instances in this matter, where Tycho Brahe's investigation can be considered as one of the most notable stories.

The Tychonic system is built using astronomical calculations which are considered the most accurate, without any involvement of telescopes. A series of observations was made, denoting the positions of the sun, the planets, and the stars. They were formulated with respect to the solar calendar. Each angular position was considered as a datum where the pattern was calculated in a twelve-month interval. Then, Brahe discovered the cycle and made a conclusion based on the pattern, i. e., the Geo-heliocentric theory.

Arguing against the idea of heliocentrism, the Tychonians insisted that if the earth was actually revolving around the sun, then there should be a position shift of the stars at some specific time within a year [14]. This argument was accepted for at least two centuries.

However, a German mathematician and astronomer namely Friedrich Bessel (1784 - 1846), discovered a way to measure the stellar parallax for a star called *61 Cygni* in the year 1838. Using a set of far more sophisticated optical tools than Brahe's, he found an evidence that had not been discovered before: a position shift of the star.

This example adds a valuable foundation to the inductive reasoning paradigm. After finding one or several patterns throughout some sets of data, conclusions are made. Further observations in the future might either strengthen the induction result, or otherwise, disprove it. Implementing a capability of making conclusions to a learning system potentially makes it more complicated. Still, this is a promising concept that has been built and conducted by the humans for thousands of years, with a potential benefit in the area of machine learning.

1.2 Background

To have an efficient learning process is one of the major interests in computer science, particularly machine learning. Investigations in this subject have been conducted for several decades under numerous flags, e. g., more recently, Autonomic Computing [17] and Organic Computing [28]. Organic Computing (OC) introduces a paradigm that a system should be equipped sufficiently to be able to develop by itself with a definite possibility of human interference in controlling the process [27]. In OC, this capability is coined with the term *controlled self-organization.*

A scenario called the *Colored Lines* [28] can be used to illustrate self-organizing behavior. As depicted in Figure 1.4, a number of lines should be sorted, following either height, color or both attributes. Starting with a random situation (see Figure 1.4(a)), each line should switch its position with one of its neighbors to achieve the team goal, i. e., forming a height ascending sequence. So, if an agent observes a taller line on its left-hand side (can also mean that the other sees a shorter teammate on the opposing side), both of them switch places. Finally, all lines are sorted according to their heights and the assignment is fulfilled (see Figure 1.4(b)). Other tasks are also possible, e. g., forming a descending sequence based on the height, or color-based grouping. Both examples are depicted in Figure 1.4(c) and 1.4(d), respectively.

Among several possibilities to design machines with a capability of performing such operation, the controlled self-organization concept plays its role as a moderate solution. OC introduces that paradigm to bridge the classical top-down and the intricate bottom-up approaches. Hence, an OC system is designed to have certain criteria of freedom, while at the same time it should also fulfill one or more objectives set by the human. Figure 1.5 provides an illustration of the concept, in relation to the colored lines task.

One way to compose a system with an ability of fulfilling such task is equipping it with an external control, as depicted in Figure 1.5(a). Such design is also called "centralized control", referring to its specific element that has the assignment to make decisions and to command every line. This top-down approach is commonly simple to design, but at the other hand difficult to cope with dynamic external changes. In conclusion, this design cannot be considered as a self-organizing system.

Figure 1.5(b) depicts another idea of a system with its own internal control, a component that independently makes decisions based on the current situation. Here, each line is considered as an autonomous agent. Instead of switching places under the command of central control, each individual observes the environment (in this case is the other agents) and takes an action. The existence of the "Controller" element within each line indicates that every

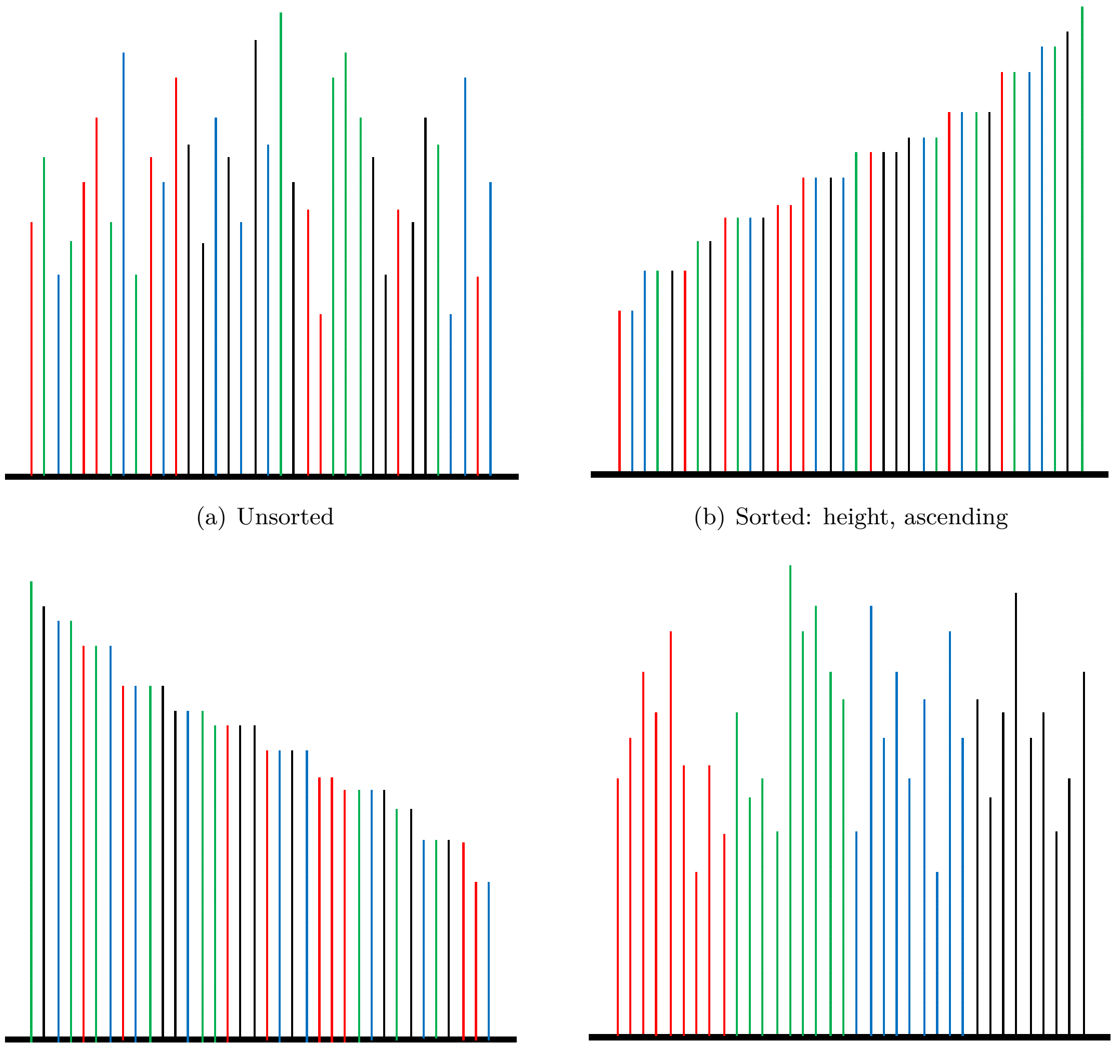
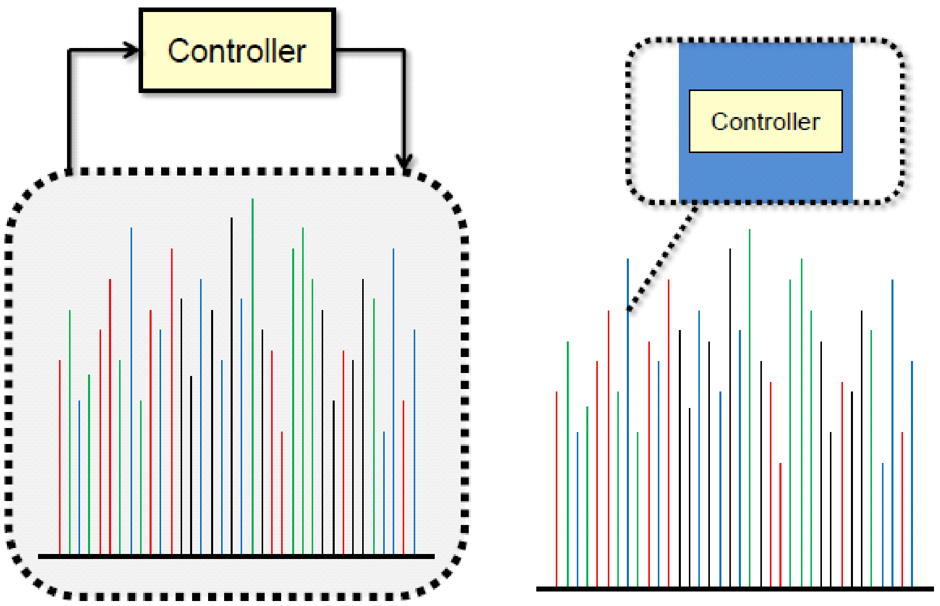

(a) Unsorted (b) Sorted: height, ascending (c) Sorted: height, descending (d) Sorted: color

Figure 1.4: Colored lines

decision is made autonomously, which remarks the idea of a controlled self-organizing system.

To realize the concept, OC employs a so-called *generic Observer/Controller architecture* [24] (see Figure 1.6(a)). The design involves a so-called System under Observation and Control (SuOC) and a learning entity comprised of an observer and a controller. After collecting the environmental data, the observer delivers some information regarding the state of the controller. Then, an action is executed as an attempt of making a specific state change at the SuOC. The environmental response after applying the decision is monitored by the observer, regarded as a feedback. Such generic concept makes further developments possible, e. g., using layers as in [8].

A desirable feedback from the SuOC indicates that the system response is appropriate. Otherwise, an unexpected or undesired behavior can be caused by one out of two possibilities: (1) the action was not chosen adequately, or (2) the observer did not provide the controller with sufficient information to make an adequate decision. A repetition of the latter may



(a) External (b) Internal

Figure 1.5: Type of controls (as in [28])

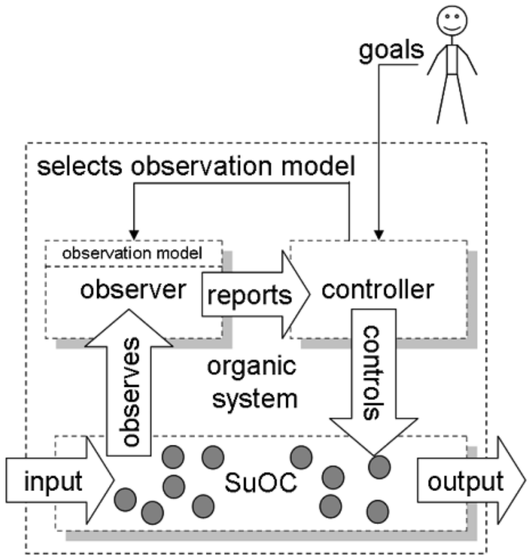
trigger an execution of "select observation model" functionality that indicates a necessity of changing the type of data obtained from the SuOC, while the former normally corresponds to the learning curve.

At its highest level, the system allows a form of external intervention coming from human or some other external entity. The user always has a full authority to modify the learning system in several ways, e. g., initiating a change in the desired SuOC state. Such design opens a chance to modify certain things according to the current needs, which can be different from time to time.

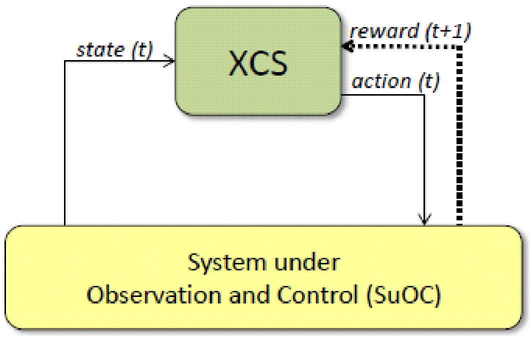
The concept introduced in this Observer/Controller architecture can be realized using various machine learning systems, e. g., Artificial Neural Network (ANN) and Accuracy-based Learning Classifier System (XCS). One of the main difference between ANN and XCS is the way of maintaining the knowledge. ANN uses sets of variables accompanying its neurons, while XCS employs rules and collects them in a population. This thesis is composed after a series of investigations using XCS instead of ANN, due to the ease of translating its population of knowledge into human language.

XCS begins each learning process after receiving the first input from the environment [35], or known as the SuOC in OC. The observed state is responded by an action that is advised by one or more rules satisfying the input. In case the received state cannot be matched by any rules (e. g., when the population is still empty), a covering mechanism occurs and an action is picked randomly. Then, the environment sends a feedback awarding the chosen action based on its suitability to the previously given state. Finally, the reward is used to update the executed rules, where the change denotes the rules' quality. Figure 1.6(b) depicts an illustration of the process.

The knowledge is kept in a population of *classifiers* (the word is used interchangeably with "rules"), where each individual contains a pair of IF-THEN parts namely *condition* and *action*. Following a relation *"IF the input matches this CONDITION, THEN execute this ACTION"*,



(a) The generic Observer/Controller architecture in OC (as in [24])



(b) XCS learns the SuOC

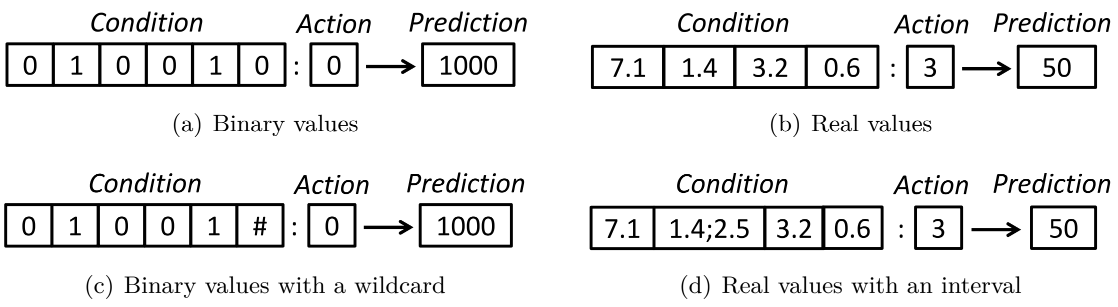
Figure 1.6: OC's learning architecture and its practical implementation using XCS

the suitability of the pair is denoted by the third attribute, i. e., *reward prediction* (will be mentioned as "prediction" afterwards). Involvement of the latter modifies the meaning of a classifier into *"IF the input matches this CONDITION, AND this ACTION is executed, THEN this is the reward PREDICTION"*. Along with those three, every rule possesses a number of book-keeping values for various calculation purposes, which will be explained in more details in Chapter 2.

Condition is the most complex part of a classifier. Quite often, it is represented in form of an array where each element is independent of the others. In the early development of XCS, this part could only contain binary values (as depicted in Figure 1.7(a)). Later, this was extended to contain ternary, alphabetical, and up to real values (as in Figure 1.7(b)). Action contains an actual decision of the system, represented by integer values. E. g., a vehicle might have five possible actions from 0 to 4, where each symbolizes "STAY" and the directions "NORTH", "EAST", "SOUTH" and "WEST", respectively. Then, prediction indicates the suitability of its condition and the action. A rule is considered accurate if the value of its prediction adequately reflects the suitability of the pair.

Since the knowledge is kept in rules, the more lessons learned also means the greater number of individuals are required. Moreover, if the system has to consider several details to make proper decisions, then the condition should be consisting of sufficient number of elements to cover them all. Such requirement increases the complexity of managing the knowledge. Also, the usage of the resources may get significantly high, making the process become less efficient.

The issue of over-populated knowledge is responded with the concept of *wildcard*. A wildcard makes a rule more general by extending the input coverage. E. g., a set of values "01001#" shown in Figure 1.7(c) can act as both "010010" and "010011" at the same time. In handling real-valued inputs, the symbolization of generality is expressed using intervals. Figure 1.7(d) depicts an example where the second element of the array is generalized, indicated by an



(a) Binary values

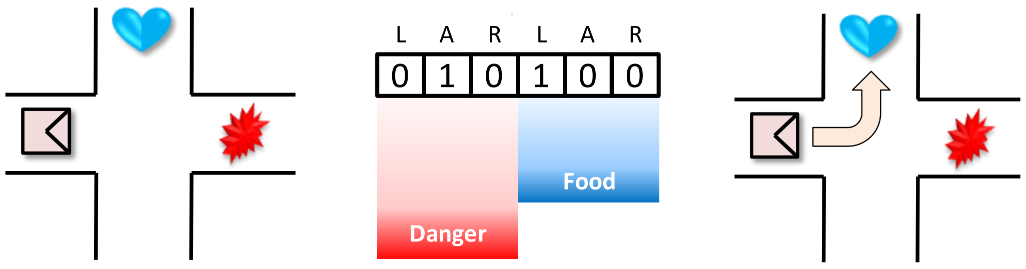
(b) Real values

(c) Binary values with a wildcard

(d) Real values with an interval

Figure 1.7: Types of classifier's condition

interval ranging from 1.4 to 2.5. The number of wildcards, as well as the width of intervals, determines the degree of generality owned by a rule.



(a) Intersection

(b) Observation result

(c) Decision

Figure 1.8: A robot should make a decision

Figure 1.8 illustrates an example of a decision making process. A robot coming from the western side of an intersection is about to choose a direction to go. The task is simple: (1) find some food; and (2) avoid dangerous obstacles. As of this moment, three options are available: (1) turn left to the North; (2) go forward to the East; or (3) turn right to the South. As shown in Figure 1.8(a), some food (symbolized by a heart) is available at the North, while an obstacle lays in the East.

Then, Figure 1.8(b) depicts the result of the robot's observation. The observed environment is coded into an array consisting of six elements. The values of the first three bits are defined by the existence of dangerous obstacles, while the others denote food availability. Each bit represents a specific direction, i. e., left (in this case is the North), ahead (the East) and right (the South), respectively. So, the first part of the observation result ("010") means *"there is a dangerous obstacle ahead"* and the rest ("100") represents *"some food is available on the left"*. Figure 1.8(c) depicts the robot's decision to turn left as expected. Hence, a full classifier representing the lesson for this situation is symbolized by ”010100 : $LEFT \rightarrow HIGH$” meaning *"IF there is a dangerous obstacle ahead AND some food is available on the left AND the action is LEFT, THEN the reward prediction is HIGH"* (in the actual rules, actions and predictions are expressed in integers and real values, respectively).

A single classifier can map the robot's situation perfectly. However, since the possible states that may occur during the learning process are numerous, an issue regarding overpopulation arises. The wildcard concept may provide the solution for this matter. Now, one very important task is left: the system should have the capability of identifying elements that can be generalized and those which must be left specific. Surprisingly, there are very few investigations on XCS that address this generalization issue very convincingly. The lack of solutions brings the idea that is pursued in this thesis, where the details are discussed in the following section.

1.3 Motivation

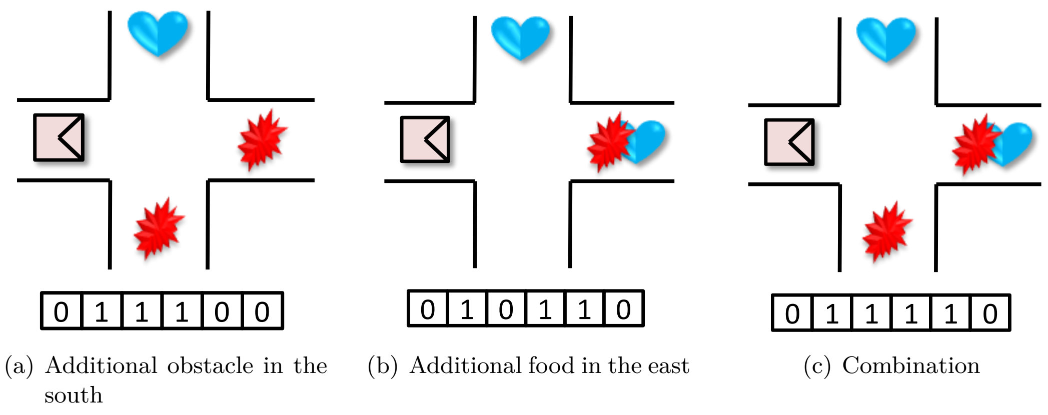
As an online learning system, XCS suits the needs of implementing the Observer/Controller architecture. It is capable of providing an action (output) directly after an input is received from the environment. Every time after a response is sent by the system, the environment delivers a reward to XCS. The feedback judges the suitability of the latest decision with respect to the previously given input. The received reward is essential for XCS to build its knowledge using a set of rules. The population is expected to contain sufficient information, while at the other hand the usage of resources should be kept minimal.

In XCS, generalization plays a central role in reducing space requirements. By disregarding insignificant (and moreover irrelevant) details, rules become more general without lowering their accuracy. Therefore, the maximal advantage of wildcards can only be achieved if the system has the capability of detecting insignificant data. Several investigations have been conducted to address this matter (e. g., [6] and [7]). Mostly, the works attempt to solve the difficulties of determining the most suitable parameter values for generalizing the rules. Some others went to the direction of modifying the system, e. g., using additional operators (as in [20]) or architectural components (e. g., as in [25]). The approach presented in this thesis goes towards this "proper generalization" direction by offering a different approach, i. e., an adaptation of the induction concept.

Referring back to the robot/intersection scenario depicted in Figure 1.8(a), an action is taken by the robot in response to a simple environmental state. In order to learn the occurring state, XCS requires at least three classifiers with varying actions: left, ahead, and right. Obviously, the highest reward would be given to the rule advocating "left". Aside from the given example, there are various situations that get optimally responded by taking the same action. Unfortunately, although the difference is slight, the robot would have to learn all states, one at a time. The learned knowledge is kept in three separate rules, making the process both space and time consuming.

Figure 1.9 illustrates some possible situations that would earn a high reward after being responded the same way with the one depicted in Figure 1.8(a). All of them have a common solution, i. e., take a left turn to the North. Without generalizations, each of the states "010100", "011100", "010110" and "011110" is assigned with three possible actions, making a total of 12 separate individuals. This means, that the system would require at least 12 learning cycles to record them all, which is considerably inefficient. Such requirement can be reduced by implementing the wildcard concept. To replace those rules, three generalized classifiers having conditions "01#1#0" and varying actions would suffice. This way, a reduction of the space requirement can be achieved without removing any learned lessons.

Now, there is a consequence of applying generalization to the classifiers. To keep the system behave appropriately, all wildcards should be placed at the correct positions. A misplaced wildcard causes the owning rule to be over-general, or in other words, containing a biased knowledge. E. g., in the previous robot/intersection scenario, placing wildcards at a direction where the danger is present (i. e., "0#01#0") is undesirable. Such condition



(a) Additional obstacle in the south (b) Additional food in the east (c) Combination

Figure 1.9: Alternate environmental situations

completely disobeys the situation at the East and obviously decreases the comprehensiveness of the knowledge. The cost is not only the time required to relearn the situation, but also some usage of space.

In the original XCS, the placement of wildcards occurs at two separate occasions, namely, while covering an input and updating the classifiers. When an input is received, the system checks whether it has been learned. Covering plays its role if the population currently has no information about the state. If that happens, XCS performs a randomization and uses a probability variable to determine whether to cover each element either using a wildcard or the original state.

Figure 1.10 visualizes how an input is covered using a 25% wildcard probability, which also means that there is a 75% possibility of putting the original value. Each red line denotes the result of a randomization. When it lies in the black area, the corresponding bit would contain a wildcard. Otherwise, the original value is used to cover the current input bit. Finally, the randomization provides a result that makes the system cover the input with "#101#0" instead of the original values "010100".

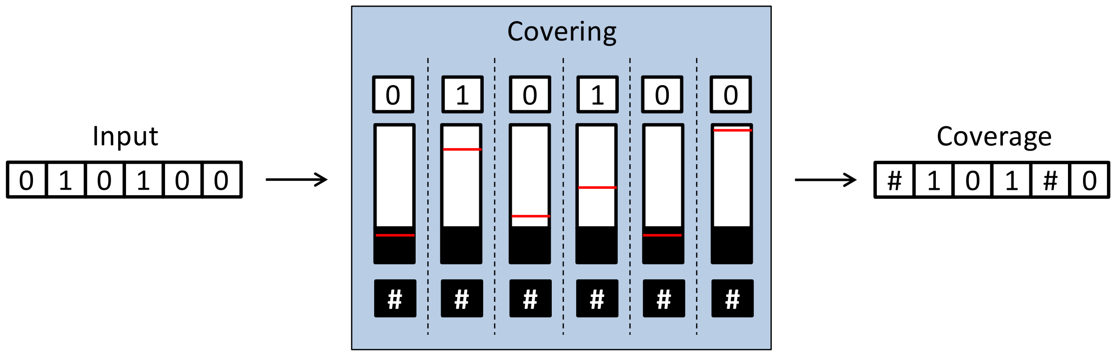


Figure 1.10: An input is covered using a 25% wildcard probability

The random process in covering has a drawback that it potentially causes some inaccuracy for the created rule. Suppose such coverage is performed by the robot in Figure 1.8, important information is lost (i. e., danger at left and food ahead). There is also an extreme possibility that the input is covered with six wildcards (i. e., "######"). Although the probability for such an "accident" can be considered very small, the coverage mechanism still allows such thing to occur.

To minimize the effect of misplaced wildcards, XCS is equipped with a rule manipulation procedure, i. e., Darwinian genetic operators. While updating the classifiers, XCS attempts to increase the quality of the rule by performing mutations and crossovers. Different from covering, these techniques consider the accuracy of each rule before selecting it as one of the parents.

Genetic operation plays its role in attempting to find good rules by creating a child based on the already existing ones. Such technique may increase the learning speed in finding favorable rules. However, some randomization is also involved here. The random process does not only affect the placement of wildcards, but also the overall learning process of the system. Moreover, employing Darwinian genetic operators also makes XCS more complicated, denoted by the number of required parameters (and also a series of experiments) in the attempts of discovering general and accurate rules.

So, although the additional method assists the system in generalizing rules, it still has a potential of slowing down the overall learning process. In this thesis, a method called *Rule Combining* (RC) is proposed to replace the approach to generalization used in the original XCS. It eliminates the negative impacts of randomizations by replacing them with a heuristic-based genetic operator. Adapted from the induction concept, RC attempts to discover general and accurate rules by making conclusions based only on the learned knowledge. More information regarding that is provided in the following section.

1.4 Contribution

This thesis describes the effects of modifications to the classifier systems. A series of simulations shows that XCS-RC is capable of providing a set of results that indicates some significant improvements compared to the original XCS. The achievement is gained by implementing three mechanisms that are designed based on lessons taught by the history of science: induction, experiment and knowledge correction.

In XCS, the general term *induction* is implemented using a name "Rule Combining". It is implemented by taking some values from a pair of parents and used them to conclude the properties of the child. Firstly, the system should identify any underlying patterns within a set of experienced rules. In case one or more are found, a generalized offspring is produced based upon them. Then, the system examines the newly created rule, looking for any contradicting knowledge. The inexistence of such disproving rule implies that the child is a valid conclusion and therefore allowed to enter the population, replacing a number of "older" classifiers.

However, there is still a possibility of performing a hasty generalization. Although an examination is run, a child might be created based on insufficient amount of information. In other words, an invalid conclusion might be accepted due to the inexistence of disprovals. Figure 1.11 illustrates this matter using a timeline. A child might pass the examination mechanism, but later afterwards, a single unlearned knowledge might dissolve the knowledge.

Two approaches are prepared to overcome the disadvantageous situation, i. e., preventive and correction. As an anticipation, the system attempts to gather more knowledge by *conducting experiments*. Every time there is a chance to gain a new lesson by picking one or more inexperienced rules, the system prioritizes such decision. This approach increases the comprehensiveness of the system's knowledge, leading to a lower chance of hastiness.

Unfortunately, simulations show that such preventive technique cannot guarantee that the

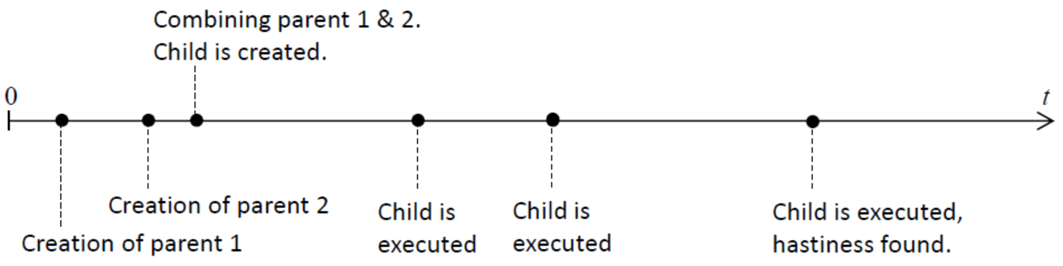


Figure 1.11: Hastiness presented using a timeline

problem is automatically solved. Some indications of producing hasty children still appear. To handle this matter appropriately, XCS-RC is equipped with a *knowledge correction* mechanism. Whenever a classifier is identified hasty, the system eliminates the invalid knowledge and replaces it with a new rule.

The concept of knowledge correction is adapted based on the history regarding the Tychonic system, where a new discovery disproved the conclusion that was made centuries before. This gives a significant contribution to the system, making its knowledge more comprehensive. Some detailed descriptions regarding these modifications are given in Chapter 3.

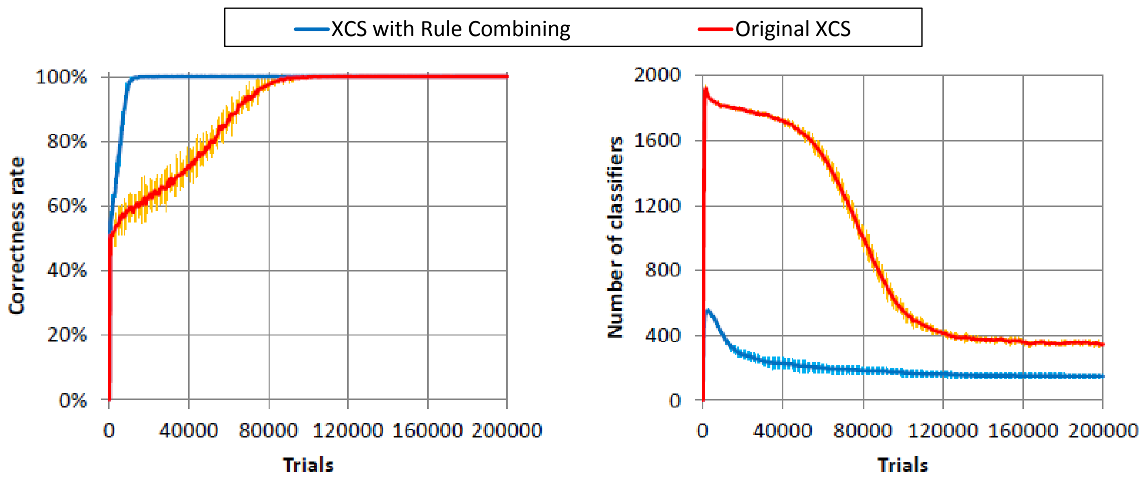
XCS-RC is equipped with the capabilities of performing inductions, experiments, and knowledge corrections. To deliver some early insights regarding the contribution, some comparisons are provided here. Graphics are presented from a number of observation data obtained from a series of simulations, where XCS-RC and the original XCS are run using an identical set of parameters and settings. A full explanation of the experiments and the corresponding results is given in Chapter 5.

A scenario called multiplexer is chosen here, due to its suitability for playing the testing role. Moreover, this task is used by Stewart Wilson as a standard XCS testing problem for handling environments consisting of both binaries (see [35, 36]) and real values (as in [37]). The judgement is determined by observing two indicators: performance and population size. The former denotes correctness rate which should be maximized, while the latter depicts the number of classifiers required to gain such result and should be minimized. In case both systems is capable of achieving those goals, the one with a quicker learning rate is pronounced as the winner.

Figure 1.12 depicts the results provided by both systems playing a multiplexer role with 20 bits of input, visualized based on an average out of 20 runs. Both systems indicate a success in performing a 100% correctness rate (see Figure 1.12(a)). However, XCS-RC reaches the expected value by requiring smaller number of learning cycles compared to the opponent. Figure 1.12(b) illustrates another advantage of employing RC, showing the ability of the proposed method in significantly reducing the population size. Therefore, it is fair to state that the proposed system is an improvement from the original XCS, where the difference highly depends on the success of induction.

1.5 Guide for Readers

For a comfortable way of reading this thesis, the writing is structured this way: the introductory chapter opens the thought by providing some philosophical foundations, a brief description of the issues and how this thesis addresses it by proposing a novel solution. Chapter 2 provides the binary-mode XCS, beginning with a flashback, followed by some detailed



(a) Performance (b) Population size

Figure 1.12: Results for 20-bit binary multiplexers

explanations about the original technique.

Then, Chapter 3 continues the discussion by presenting the proposed RC method, brings up the novel improvement of XCS which reconstructs the basic mechanism of its rule discovery functionality. Following afterwards, Chapter 4 presents the XCSR, a real-valued mode XCS, along with some modifications of XCS-RC to add the capability of handling such problems.

To evaluate the proposed method, several comparisons between XCS (and some of the variants) and XCS-RC for binary and real values are described in Chapter 5, using a number of known scenarios. Chapter 6 introduces the capability of the proposed system in performing feature selection, which can only be obtained from classifier systems with a powerful generalization capability. Finally, Chapter 7 summarizes this thesis by presenting some conclusions and foreseen outlooks for future works.

1.6 Acknowledgement

The conducted investigations and the writing of this thesis are financially supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) within the priority program DFG-1183 Organic Computing. Some of the works have been published in a close cooperation between the research group of Prof. Dr. Hartmut Schmeck from Karlsruhe Institute of Technology in Karlsruhe, Germany and the group of Prof. Dr.-Ing. Christian Müller-Schloer from Leibniz Universität in Hannover, Germany.

CHAPTER 2 ACCURACY-BASED LEARNING CLASSIFIER SYSTEM

"He who learns but does not think, is lost. He who thinks but does not learn, is in great danger."

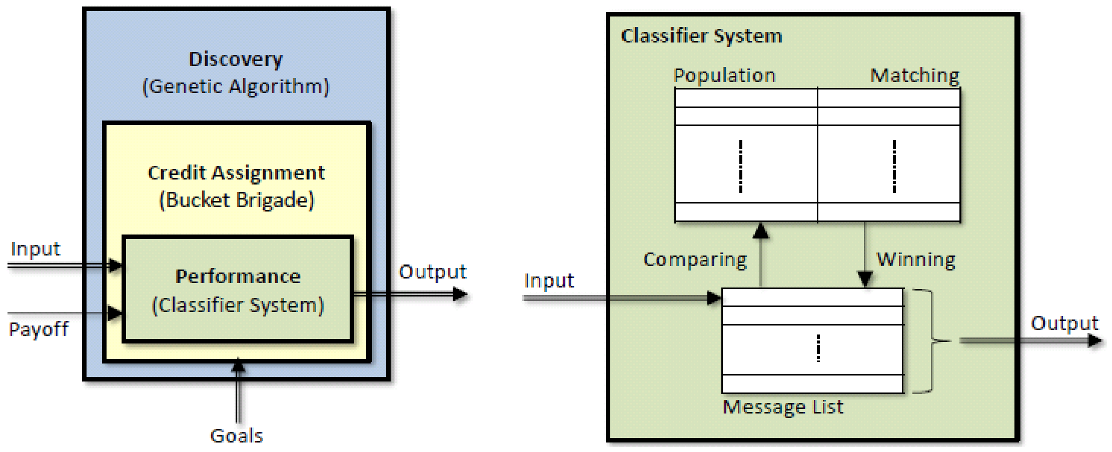
(Confucius)

Nowadays, machines play a major role in assisting humans, either directly in daily activities or indirectly such as through industrial productions. This involves the capability of machines in performing various learning processes, especially in handling complex problems. The advancement of technology brings the world to a direction that requires an intelligent behavior performed by computers and gadgets. Within the scope of answering such challenge, this thesis proposes a novel paradigm of machine intelligence, derived from the concept of induction. Its aim is to improve a well-known machine learning technique, *Accuracy-based Learning Classifier System* known as *XCS*, a variant of *Learning Classifier System* (LCS).

LCS was introduced by John H. Holland in 1976 as an online learning method that maintains a number of rules called *classifiers*. Each individual in the population is equipped with a pair of attributes namely *condition* and *action*, performing an "IF-THEN" functionality. The suitability of the action in responding to the condition is expressed by the third attribute, namely *reward prediction* (commonly mentioned as *prediction*). LCS retrieves inputs from the environment in form of an array of binaries, where each bit represents a specific meaning. The input length follows the number of observed states in the environment.

There are three functionalities employed in Holland's LCS [2], i. e., discovery, credit assignment and classifier system itself. As seen in Figure 2.1(a), the environment is observed by an input interface and the result is collected in a so-called *message list*. After receiving a number of inputs, the list is compared to another set called *rule list* (or rule base). Classifiers in the rule list that satisfy at least one input are grouped based on their actions. After choosing an output, rules advocating the winning action are collected in a specific set. This collection becomes the new message list and replaces the old one (see Figure 2.1(b)).

The chosen action is delivered to an output interface. When the payoff is received, the credit assignment updates the attribute indicating the quality of the winning rules using the Bucket Brigade algorithm. At the other part of the system, genetic operators attempt to pick some good rules before processing them in order to obtain some better ones. In this initial version of LCS, the quality of a rule is solely determined by the value of the payoffs it receives.



(a) Organization

(b) Basic parts

Figure 2.1: Holland's LCS as explained in [2]

Shortly afterwards in 1978, an improvement of LCS was introduced by Holland and Reitmann (in [15]). Here, a so-called *fitness* attribute is employed to denote the accuracy of a classifier in predicting the external feedback [3]. So, in this improved version of LCS, a lower reward does not indicate that the corresponding rule is less useful. Instead, it represents the quality of the classifier in predicting the payoff. This point remarks the first modification of LCS and becomes the basis of the accuracy-based variant introduced by Wilson in 1995.

2.1 Overview

XCS is oftenly considered as one of the most successful variants of LCS. Commonly starting with an empty knowledge, a learning agent attempts to behave intelligently in responding to every observed environmental state. The knowledge is gained by obtaining judgements from the environment in form of quantitative rewards, denoting the suitability of the system response to the received state. A high feedback value implies that the response is suitable, a higher one means better, while a lower payoff indicates that the chosen action is less desirable.

Figure 2.2 depicts the mechanism of XCS in schematic representation. It retrieves input in form of an array where each element contains a binary value, either "0" or "1". Every time a state is received, the system executes a specific decision making procedure to pick one of the available responses. The number of possible actions is defined by the user, symbolized by a.

The learning process starts when the system receives the first environmental state. When an input is received, a matching attempt is conducted. Here, the system examines the *population of rules* (or population, commonly symbolized by $[P]$) and collects classifiers that satisfy the input in a so-called *match set* $[M]$. In case the number of available actions in $[M]$ is less than a (including when the number of matching rules is zero), a *covering* mechanism is executed. XCS composes one or more new classifiers and sets their condition attributes based on the input. The newly created rules are stored into $[P]$ and collected in $[M]$ at the same time.

In the match set, XCS performs a so-called *action selection* step to choose one of the available outputs. The process of picking a winning action is executed either using exploitation or exploration technique. Exploiting means that the system deterministically chooses an ac-

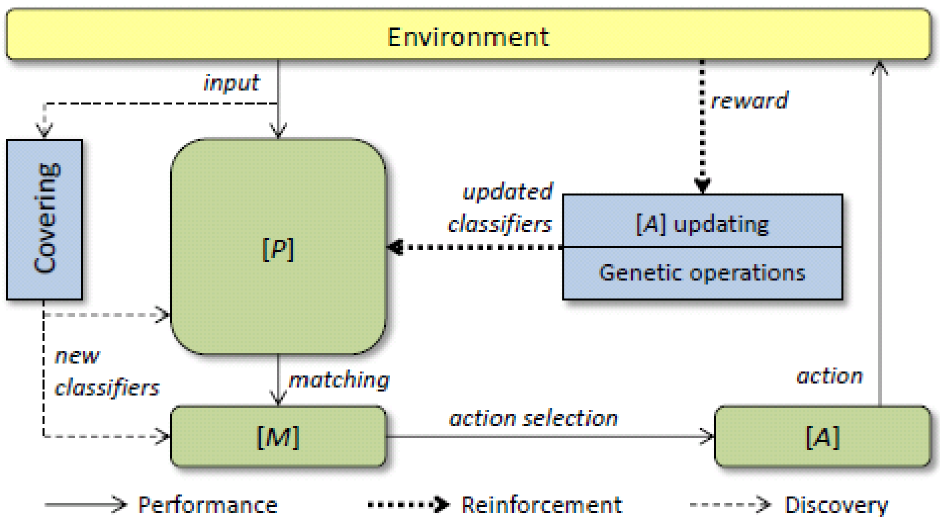


Figure 2.2: XCS full cycle

tion with the highest predicted payoff, while exploring is commonly used to examine other options. Then, one or more rules advocating the winning output enter a so-called *action set* $[A]$, and the response is sent to the environment. Afterwards, the system prepares to receive a feedback.

The environment judges the quality of the chosen action according to its suitability in responding to the previously given state. Then, XCS uses the received feedback to update the classifiers in $[A]$. If the output is desirable, then the payoff tends to be high. Otherwise, a lower feedback is sent. This part is called a *reinforcement* step, executed following the *Q-Learning* method proposed by Watkins [33]. The result of the update is calculated by weighing the old value and the incoming reward. Finally, the learning cycle closes by restoring the updated rules to $[P]$.

There are three main components of XCS, i. e., performance, reinforcement and discovery. Each part plays its role differently, despite the strong connection among them. The performance component is assigned with the task of responding to the input with an appropriate output. It checks the population for matching classifiers, stores those rules to the match set, picks a winning action and executes it. The reinforcement part is implemented through rule updates. After receiving the payoff from the environment, XCS calculates the new values of the classifier's attributes accordingly.

The third component, discovery, is responsible to produce general and accurate rules. Generality denotes the capability of the rule in covering numerous states, while accuracy indicates the suitability of its prediction in reflecting the condition-action pair. A rule that is capable of covering several states but at the same time inaccurate is expressed using a term *over-general*.

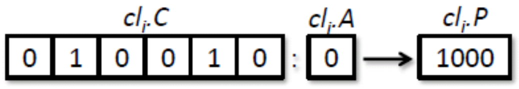
In the implementation, the discovery component consists of four techniques, i. e., covering, genetic operation, subsumption and deletion. As previously explained, covering has an assignment to create rules at the early stage of the learning cycle. After some time, the population commonly consists of a great number of classifiers. When the population size exceeds a user-defined threshold N, the deletion plays its role by removing some rules having low accuracy from $[P]$. The genetic operation attempts to create some good rules by manipulating the condition values of some parent classifiers, while subsumption has the task to eliminate any indications of rule redundancy. Now, to provide a clear description regarding the system, the following section explains each attribute owned by a classifier.

2.2 Classifier

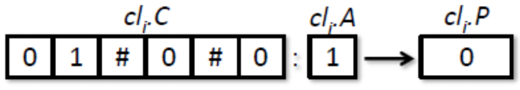
XCS stores its knowledge in form of classifiers and collects them in $[P]$. Each rule is equipped with a set of attributes. Among those, the main roles can be expressed as an "IF", a "THEN" and a value indicating the suitability of the pair. The "IF" part is *condition*, commonly symbolized with $cl.C$, an array whose elements are independent of each other. Different from that, the "THEN" part named *action* ($cl.A$), is commonly expressed by an integer. Although, basically, any expressions denoting an output are also suitable (e. g., "LEFT" or "STOP").

The pairing condition-action guides the system to respond to the received states. So, the third attribute named *reward prediction* (commonly mentioned as *prediction* and symbolized by $cl.P$) is preserved to keep the suitability judgement of the pair. If $cl.A$ is a desirable response to one or more states covered by $cl.C$, then the value of $cl.P$ would be relatively high. Otherwise, an unsuitable action in responding to the condition is indicated by a low prediction value. Hence, the relation involving the three main attributes can be expressed using a sentence *"IF the input matches cl.C AND the action is cl.A, THEN the reward prediction is cl.P"*.

Figure 2.3(a) illustrates a common way to symbolize a classifier, i. e., by using the notation "$cl.C : cl.A \rightarrow cl.P$" (e. g., "$010010 : 0 \ \rightarrow \ 1000$"). In the early development of XCS, each element in the condition could only contain either a binary (i. e., "0" and "1") or a wildcard ("#"). Later afterwards, a number of variants was developed to handle various input types, e. g., continuous values [37].



(a) Three main attributes owned by cl_i



(b) $cl_i.C$ with two wildcards

Figure 2.3: A classifier cl_i

A wildcard expresses the generality of a particular bit. Since a "#" plays both roles as "0" and "1", a generalized classifier is capable of matching more than a state. Each wildcard multiplies the rule's coverage by two, e. g., a classifier cl_i depicted in Figure 2.3(b). Such rule is able to match four possible inputs (i. e., "010000", "010010", "011000" and "011010"). Any appearance of those states leads cl_i to entering the match set, with a further chance of being chosen to enter the action set.

After an action is executed, classifiers in the action set are updated following a received payoff P. Then, the system calculates new values of three attributes, i. e., $cl.P$, *fitness* $cl.F$, and *prediction error* $cl.\varepsilon$. The $cl.P$ is updated following a formula adapted from the Q-Learning technique (more details in Section 2.5). As a usefulness indicator, $cl.F$ denotes a classifier's relative accuracy in comparison to the other $[A]$ members. When the size of $[P]$ exceeds the threshold, the value of $cl.F$ becomes a decisive factor for the rule deletion process. Then, $cl.\varepsilon$ is assigned with $\sum |P - cl.P| \div cl.exp$, where $cl.exp$ refers to the rule's experience, denoting how many times it enters $[A]$.

To anticipate two or more classifiers having similar attribute values, XCS introduces the so-called *numerosity* attribute $cl.num$. Its value expresses the number of identical rules in $[P]$. In performing the learning cycles, some discovery processes (e. g., Darwinian genetic operation) may create rules having the pair condition-action whose values are identical to an existing classifier. When this occurs, the system keeps only one of the classifiers and eliminates the others. To compensate the missing rules, the value of $cl.num$ owned by the survived one is incremented by the number of eliminated individuals.

A classifier having $cl.num > 1$ is called a macro-classifier. The opposite, i. e., each individual contained by a macro-classifier, is a micro-classifier. When a rule cl_i with $cl_i.num > 1$ is chosen to be removed (e. g., when the population size exceeds the allowed amount), the system simply decrements $cl_i.num$ by one. Contrary to that, if a deleted classifier cl_j having a numerosity $cl_j.num = 1$, then cl_j is completely removed from the population. As a summary, Table 2.1 provides the list of attributes owned by a classifier.

Table 2.1: Attributes of a classifier

Symbol	Name	Description
$cl.C$	Condition	Plays the "IF" role
$cl.A$	Action	Plays the "THEN" role
$cl.P$	(Reward) prediction	Suitability judgement of the "IF-THEN" pair
$cl.F$	Fitness	Usefulness level
$cl.\varepsilon$	Prediction error	Average of difference between $cl.P$ and the payoff
$cl.exp$	Experience	Counter for entering $[A]$
$cl.num$	Numerosity	Number of identical classifiers

The population of rules represents XCS's knowledge, i. e., the learned lesson from the environment. Since XCS is designed to solve a wide variety of problems, a set of user-defined parameters is provided. An adequate set of parameter values leads XCS to perform optimally in reaching the assigned goal. Therefore, a section focusing on the discussion of the system parameters is provided as follows.

2.3 System parameters

A learning system should be designed to face various levels of tasks, from a simple problem to the complex ones. Based on such idea, a set of user-defined parameters is provided for adjustment purposes. Assigning appropriate values to those variables leads the system to its optimal performance. Table 2.2 contains a list of XCS's system parameters, summarized from [35, 36] and [4]. For ease of understanding, the list is presented in five groups indicating the parameter's assignment or its contribution to the learning process.

The first group "Initialization" consists of three variables. They determine the default values of some classifier's attributes when the rule is created. Firstly, the value of p_I remarks the initial reward prediction $cl.P$. The prediction error $cl.\varepsilon$ follows ε_I, while F_I becomes the fitness value $cl.F$. In most XCS variants, the values of these parameters have very small effect to the learning process, if not none. This is due to the mechanism employed by XCS, which commonly gives very small role to the initial settings by not involving them in further calculations.

The four parameters grouped into "learning and fitness evaluation" play their role in a central part of the learning process. The values determine how XCS calculates the new

Table 2.2: XCS parameters

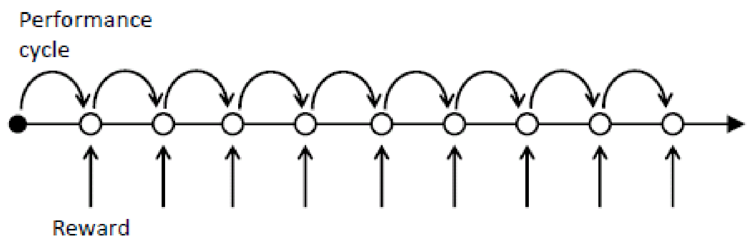
Group	Symbol	Name / Function
Initialization	p_I	Classifier's reward prediction
	ε_I	Classifier's prediction error
	F_I	Classifier's fitness
Learning and fitness evaluation	α	Accuracy distinction rate
	ν	Power function
	ε_0	Accuracy threshold
	β	Learning rate
Discovery	$P_\#$	Wildcard covering probability
	θ_{GA}	GO threshold
	pX	Crossover probability
	pM	Mutation probability
	(unsymbolized)	GO prediction error reduction
	(unsymbolized)	GO fitness reduction
Population size reduction	N	Maximal population size
	δ	Fitness threshold for deletion
	θ_{del}	Deletion threshold
	θ_{sub}	Minimal experience to subsume
Multi-step	γ	Discount rate (for multi-step)
	(unsymbolized)	Teletransportation (for multi-step)

fitness of a classifier after receiving a payoff. More details regarding this matter are discussed in Section 2.5. Specifically for β, it is also involved in determining a "preparation time" for new rules, before a "normal" calculation is applied to them.

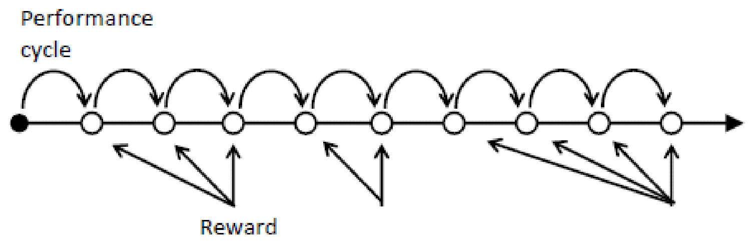
Six parameters are included in the "discovery" group, where five of them are related to the genetic operation (the term "Genetic Algorithm" or "GA" is used to preserve the original name). There are two unsymbolized parameters that are only used in the algorithmic implementation (source code). Meanwhile, the parameter $P_\#$ does not have any relation to genetic processes. It expresses a probability variable for determining whether an element of the input is covered using a wildcard or the original value. A higher value of $P_\#$ indicates a greater chance of performing generalization in covering.

Continuing to the "population size reduction" group, each of the four parameters in this category plays specific role in performing subsumption and deletion. While the latter means eliminating a micro-classifier or subtracting a macro-classifier's numerosity by one, the former refers to deleting a rule and storing its attribute values to the subsuming classifier. A subsumer is always more general than the ones being subsumed. E. g., a rule "01#" : $0 \rightarrow 100$ is allowed to subsume other classifier having a condition either "010" or "011", as long as they have the same action. Then, the parameter N denotes the maximally allowed number of classifiers in $[P]$. When the population size exceeds N, XCS performs a deletion process by picking some rules and removing them from the population (or decrementing the numerosity).

The two parameters in the last group are preserved for *multi-step* tasks. Figure 2.4(a) and 2.4(b) illustrate the difference between the single and the multi-step operations. While the former always receives a payoff directly after executing an action, the latter regulates the reward mechanism differently. In a multi-step task, the payoffs are received after a series of steps. A learning cycle closes when a specific condition is met. The environment sends



(a) Single-step: directly after completing a cycle



(b) Multi-step: distributed to a number of previous steps

Figure 2.4: Reward assignment in single and multi-step problems

the feedback and the system distributes the payoff to all uncredited classifiers which were executed.

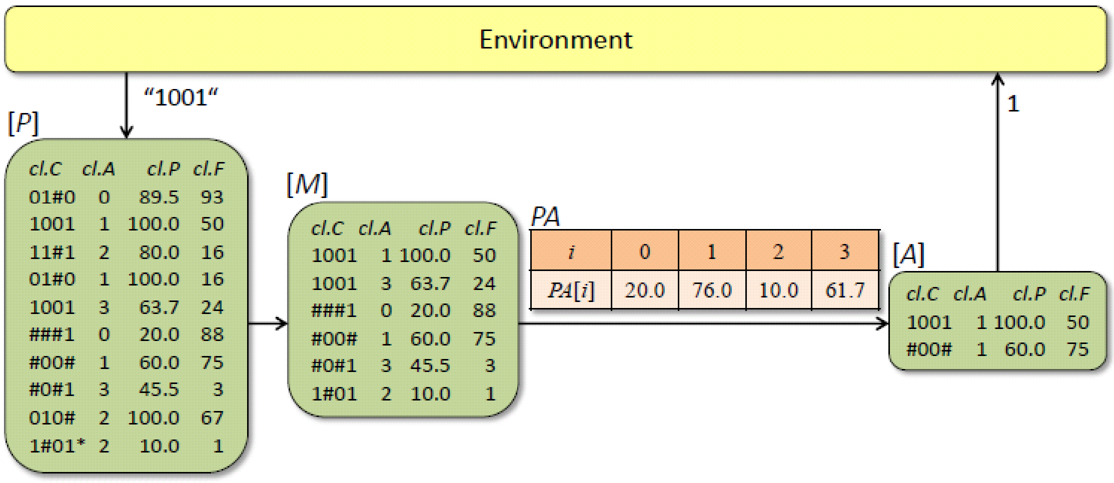
Finally, an unsymbolized variable namely *teletransportation* is prepared as a threshold for the number of steps. XCS uses this parameter as a way to terminate the current process and to close a learning cycle. A suitable example for this type of scenario is provided in Subsection 5.2.4, where a virtual entity called "animat" attempts to find the quickest way to some food. The reward is given once the food is "eaten", or when the animat keeps wandering around until the number of steps exceeds the value of *teletransportation*. Then, the last parameter in this group is γ. It denotes a discount rate for the multi-step scenarios, which is handled differently to the single-step tasks.

A suitable set of parameter values enables the system to learn the given problem properly. A good performance means that every received state is responded appropriately, denoted by the highest possible reward. In processing every input, the system involves three components, i. e., performance, reinforcement and discovery. Each of them has a specific job, as discussed in the following section.

2.4 Performance component

The term *performance component* refers to a straightforward process involving the population $[P]$, the match set $[M]$ and the action set $[A]$. This part is also responsible for all activities connecting those sets. When the system receives an input, the state is compared to all classifiers in $[P]$. Matching individuals enter $[M]$, where one of the available outputs is chosen in the action selection step. Then, one or more rules advocating the winning output enter $[A]$ and the chosen response is executed.

Figure 2.5 depicts an example where XCS having four possible outputs (i. e., 0 to 3) receives an input $x =$ ”1001” from the environment. Since none of five matching classifiers in $[P]$ advocates "2", a new rule $cl_{10} =$ ”1#01 : 2 $\rightarrow$ 10.0” is created (marked with an asterisk). The initial values of $cl_{10}.P$ and $cl_{10}.F$ follows the user-defined parameters P_I and F_I, respectively.



[P]

cl.C	cl.A	cl.P	cl.F
01#0	0	89.5	93
1001	1	100.0	50
11#1	2	80.0	16
01#0	1	100.0	16
1001	3	63.7	24
###1	0	20.0	88
#00#	1	60.0	75
#0#1	3	45.5	3
010#	2	100.0	67
1#01*	2	10.0	1

[M]

cl.C	cl.A	cl.P	cl.F
1001	1	100.0	50
1001	3	63.7	24
###1	0	20.0	88
#00#	1	60.0	75
#0#1	3	45.5	3
1#01	2	10.0	1

PA

i	0	1	2	3
$PA[i]$	20.0	76.0	10.0	61.7

[A]

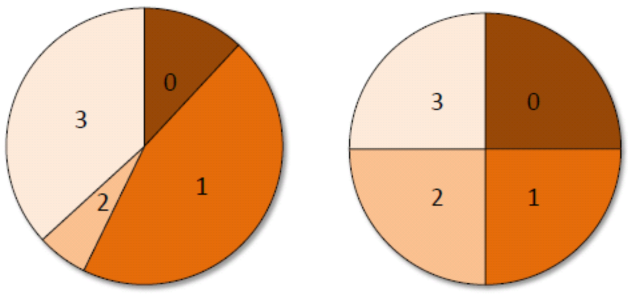
cl.C	cl.A	cl.P	cl.F
1001	1	100.0	50
#00#	1	60.0	75

Figure 2.5: Performance component of XCS

All six individuals that satisfy the input are collected in $[M]$, including the newly created cl_{10}. Then, a so-called *prediction array* (PA) is composed in four elements, following the number of possible actions. Each $PA[k]$ contains an average of the fitness-proportionate value of all $cl_i.P$ where $cl_i.A = k$, using a formula depicted in Equation 2.1.

$$P_i = \frac{\Sigma cl.P \times cl.F}{\Sigma cl.F} \tag{2.1}$$

In general, two options are available to determine the winner, i. e., exploitation and exploration modes. The former assigns XCS to pursue the most advantageous payoff by picking the highest $PA[k]$ value and executes the action k. Figure 2.5 depicts XCS's effort to maximize the feedback by choosing the action "1", following the highest value owned by $PA[1]$. In case more than one PA members currently have the highest value, XCS picks one of them randomly.



(a) Roulette wheel (b) Uniform

Figure 2.6: Probabilities in explorative action selection

Different to that, doing an explorative selection means that the system is allowed to select any available action. One of the aim in using this mode is to observe the suitability of an output to a specific state that is currently received. Figure 2.6 displays the graphical representation of the probabilities based on the case depicted in Figure 2.5. The roulette wheel is composed using the proportions from the set of values in PA. This means, a better action has a greater probability to be picked. Different to that, the uniform probability distribution simply gives all options an equal opportunity to be chosen.

After an action is selected, the corresponding classifiers enter $[A]$ and the output is fired. Then, the environment sends a feedback to the system as a judgement of the chosen output. The reward is used by XCS for calculating some attributes owned by the classifiers in $[A]$. This is a part of the reinforcement component, and the details are provided in the following section.

2.5 Reinforcement component

Feedback is one of the key factors in determining the success of XCS in learning the environment. By examining the payoff P, XCS discovers the suitability level of the condition-action pair owned by the chosen classifiers. A high reward means that the action is a desirable response to the received state, while a low one expresses its unsuitability. Then, XCS calculates and updates the values of $cl.P$, $cl.F$, and $cl.\varepsilon$ owned by the rules in $[A]$. The updated classifiers return to the population and this moment remarks an additional lesson learned by the system (see Figure 2.7).

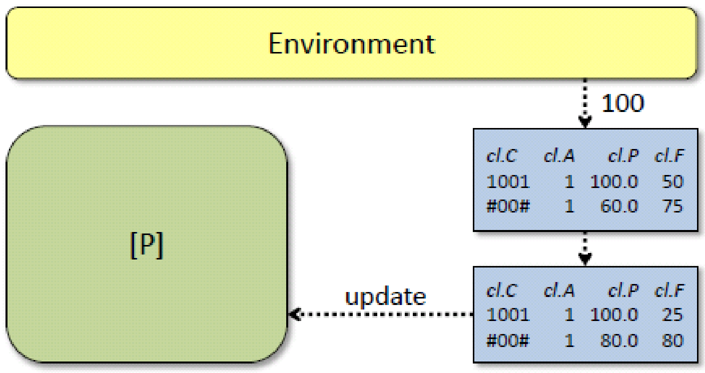


Figure 2.7: Reinforcement component of XCS

The updates of prediction $cl.P$ and prediction error $cl.\varepsilon$ follow two formulas provided in Equation 2.2 and 2.3, respectively. The determining variables in the process are the classifier's experience $cl.exp$ and the parameter β. Both equations indicate that XCS does not apply "normal" calculation to new rules. But instead, it minimizes the effect of the initial values by employing some preparatory phase. Different to the ones applied to $cl.P$ and $cl.\varepsilon$, the calculation for $cl.F$ is slightly more complicated.

$$cl.P = \begin{cases} \dfrac{cl.P \times (cl.exp - 1) + cl.P}{cl.exp} & \text{if } cl.exp < \dfrac{1}{\beta}, \\ \\ cl.P + \beta \times (P - cl.P) & \text{otherwise} \end{cases} \tag{2.2}$$

$$cl.\varepsilon = \begin{cases} \dfrac{cl.\varepsilon \times (cl.exp - 1) + |P - cl.P|}{cl.num} & \text{if } cl.exp < \dfrac{1}{\beta}, \\ \\ cl.\varepsilon + \beta \times (|P - cl.P| - cl.\varepsilon) & \text{otherwise} \end{cases} \tag{2.3}$$

The usefulness rate of a classifier cl_i is indicated by the value of $cl_i.F$. A closer value to 1.0 means a higher chance of survival against deletions owned by the corresponding classifier. XCS updates the fitness attribute by involving all rules in $[A]$, starting with a calculation for *accuracy* κ_i belonging to each classifier cl_i, as depicted by Equation 2.4. Then, it continues to determine the value of each rule's *relative accuracy* κ'_i (see Equation 2.5).

$$\kappa_i = \begin{cases} \alpha \times \left(\dfrac{cl_i.\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{if } cl_i.\varepsilon \geq \varepsilon_0, \\ 1 & \text{otherwise} \end{cases} \tag{2.4}$$

$$\kappa'_i = \frac{\kappa_i}{\Sigma \kappa_i} \tag{2.5}$$

$$cl_i.F = \begin{cases} cl_i.F + \beta(\kappa'_i - cl_i.F) & \text{if } cl_i.exp \geq \dfrac{1}{\beta}, \\ \dfrac{\kappa'_{i,t-1} + \kappa'_{i,t}}{2} & \text{otherwise} \end{cases} \tag{2.6}$$

Finally, the value of each $cl_i.F$ is defined using Equation 2.6. One might say that by determining $cl.F$ this way, its value does not actually reflect a true fitness of a classifier. The argument is based on the fact that the process only involves classifiers that are currently entering $[A]$. Addressing this matter, Wilson argues that such calculation would lead rules with higher fitness to be included in the discovery processes, i. e., genetic operations [35]. This way, the relativity of $cl.F$ pushes the rule to have more offspring and to stay survive. To provide the full description on the discovery component, a detailed discussion is presented as follows.

2.6 Discovery component

The discovery element is responsible for producing desirable classifiers, expressed by their generality and accuracy. Technically, this component has a task to put a sufficient number of wildcards at proper positions in a classifier's condition. A success of doing this assignment leads to a reduction of the population size. This point remarks the solution offered by this thesis,presenting a novel way of performing generalizations, i. e., by implementing the concept of induction.

Highly qualified classifiers enable the system to maximize its learning rate, as well as to minimize the occupied storage for keeping the knowledge. This aim has been targeted by numerous works, although very few of them were able to achieve both satisfying and convincing results. Butz et al., attempted to approach the solution by identifying and analyzing the evolutionary pressures in XCS [7], thereby supporting the theoretical understanding of Wilson's hypothesis. In [20], Lanzi provided a number of insights on the system configuration, while some other works attempt to identify the available rooms for further improvements (e. g., by modifying the process of deletion [18] and the selection schemes [6]). All of them successfully contributed to the development and improvement of XCS, while this thesis proposes another option using a very different approach.

Before continuing the discussion further, a detailed description regarding the techniques in the discovery component of XCS is presented as follows:

Covering Covering occurs for an input x if the classifiers in $[M]$ represent less than a predefined number of different actions. During covering, additional matching rules are created. Depending on the value of $P_{\#}$, wildcards are inserted in a new classifier's condition so it matches a set of inputs including x. The chance of having a wildcard between the bits are independent of each other, making it possible to cover the input as it is, or even having all bits containing wildcards (see Figure 2.8). In creating a rule, the values of its prediction, prediction error, and fitness are initialized with the defaults, i. e., the parameters P_I, ε_I and F_I, respectively.

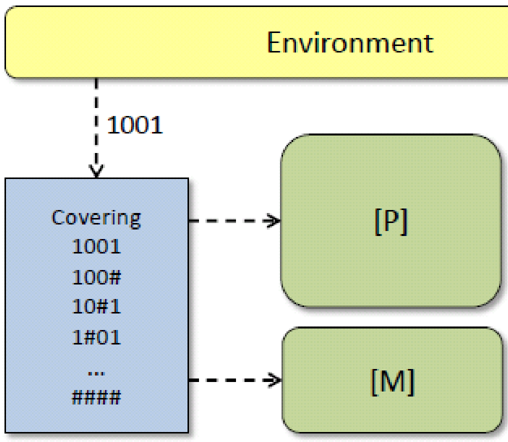


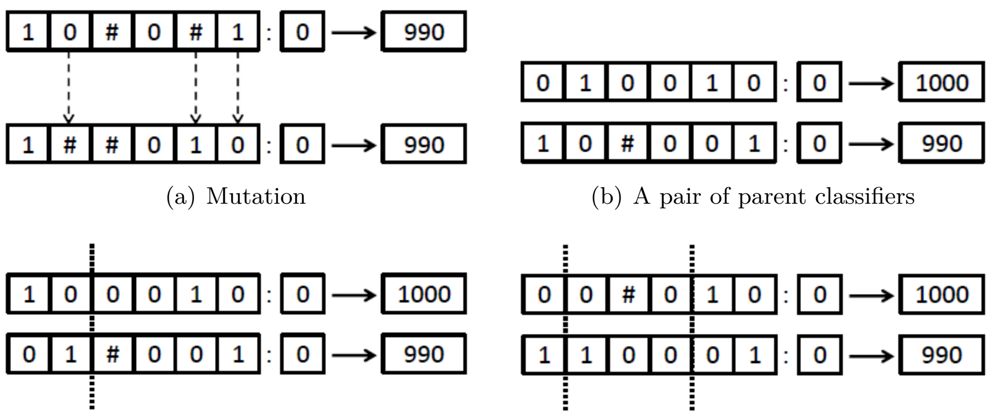
Figure 2.8: Covering an input

Genetic operators Crossovers and mutations are applied to classifiers in the action set if the average time since their last genetic modification exceeds a threshold θ_{GA}. In this case, parent classifiers are selected using fitness-proportionate [35,36] or tournament [6] selection. From the selected parents, a number of offsprings are created using mutation and crossover, as shown in Figure 2.9.

Subsumption When inserting newly created classifiers into $[P]$ or when building $[A]$, subsumption may occur. A rule cl_i may be subsumed by cl_j if $cl_i.A = cl_j.A$ and cl_j is sufficiently accurate (with respect to $cl_j.\varepsilon$) and experienced (denoted by $cl_j.exp \geq \theta_{sub}$). If so, cl_i is deleted and its numerosity $cl_i.num$ is added to $cl_j.num$.

Deletion To keep the population within its allowed size N, the system deletes a number of rules when the threshold is exceeded. A classifier is selected for deletion with a probability that is proportional to the average size of the action sets it was part of. This way, classifiers in well covered environmental niches have a higher probability to be removed. Furthermore, experience and fitness of a classifier influence its deletion probability. If the rule is experienced and its fitness is lower than the population's average fitness, its probability for being removed increases.

The attempt of finding general and accurate rules is conducted by modifying the condition. At the beginning of each learning cycle, covering involves the probability parameter $P_{\#}$ in



(a) Mutation (b) A pair of parent classifiers

(c) One-point crossover (d) Two-point crossover

Figure 2.9: Mutation and crossovers

deciding whether to put a wildcard or not. Then, Darwinian genetic operations have a chance to modify some selected rules. The modification is expected to correct any misplaced wildcards. Thus, the genetic process comes up with some rules having a more adequate condition.

However, evaluating a random process using a probability factor is difficult. Several efforts have been done (e. g., by Butz et al. [7]) to give some guidance in determining the appropriate values for the parameters $P_\#$ and those belonged to Darwinian genetic operations. Some investigations successfully assist the system to perform better, but the issue yet remains live. In conclusion, further developments are still very much expected.

In the following chapter, a novel way of improving XCS's capability is introduced under the name XCS-RC, which stands for *XCS with Rule Combining*. As mentioned previously, its technique to discover general and accurate classifiers is derived from the concept of inductive reasoning. Along with several characteristics inspired by numerous historical scientific processes, the proposed system manages to improve the learning behavior, imitating human in making conclusions.

CHAPTER 3 XCS WITH RULE COMBINING

"I am turned into a sort of machine for observing facts and grinding out conclusions."

(Charles Darwin)

Humans learn from the experience. They observe things, record and keep the results in some part of the brain. They also act, responding to their environmental state by choosing one of the available options. Then, in some cases, they retrieve feedbacks. At this point, when they consider the response as a lesson, they learn. After a number of learning experience, they have a chance to discover a pattern laying in a series of occurring events. And finally, for those who succeed in taking advantage of such opportunity, they made a conclusion based on the experience. Humans build their *knowledge*, which leads them to something known as *science*.

As a machine learning technique, XCS operates in a similar way to some of those activities. The learning system observes the environment, keeps the states, chooses a response, and finally obtains the feedback of the action. As such process runs continuously, XCS builds its very own knowledge. In the practice, after performing the learning cycles several times, some contradictions might appear. Later experience may strengthen, or contrary to that, weaken the existing knowledge. This is the way XCS updates the knowledge, with some expectations of obtaining a better accuracy.

Now, a specific point differentiating the learning process conducted by XCS and humans is the part where humans make their conclusions. The original XCS does not operate any techniques aiming to discover any possible underlying patterns within the knowledge. Its attempt to gather new knowledge is implemented through two occasions, i. e., covering and Darwinian genetic operation. Contrary to that, humans build their knowledge by making conclusions from a set of observed facts.

This thesis proposes a novel technique called *Rule Combining* (RC). It allows a machine to search deeply into the heap of knowledge, and then to build a number of conclusions based on the available information. The method is adapted from the induction concept, and is inspired by several historical scientific processes. By employing RC, the learning system performs a specific technique to find patterns, aiming for a better performance and a shorter learning time. From this point on, the proposed system is mentioned as *XCS with Rule Combining* (*XCS-RC*).

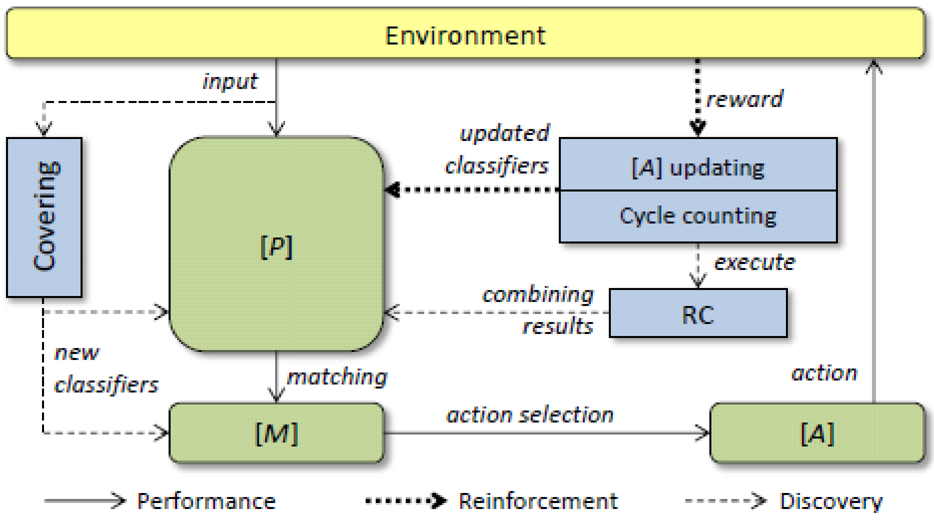


Figure 3.1: Learning cycle in XCS-RC

Figure 3.1 depicts a modified schema of XCS's learning cycle. An additional process is added after the update of $[A]$, to express the moment when RC operates the generalization attempt. The arrows going to and from the RC square indicate that the additional technique is a part of the discovery component. As a result, RC produces new rules representing a set of conclusions based on the available knowledge. Similar to the Darwinian genetic operations in the original XCS, modifications are focused on the condition attribute of a classifier ($cl.C$). However, XCS-RC does not employ Darwinian genetic operations in any part of the learning activities since it is replaced by RC.

3.1 Heuristic-based Evolution

Performing a genetic-based manipulation to the rules is possible since the attribute $cl.C$ is an array consisting of independent values. Therefore, implementing Darwinian evolution has a good chance to create better rules. Mutation and crossover select individuals that fulfill certain quality criteria to create some offsprings, which are expected to be better individuals. At the other hand, RC introduces a novel heuristic approach for discovering general and accurate rules. Different from those employed by XCS, this method does not implement any form of randomization. Instead, RC attempts to find better rules by solely performing deterministic genetic procedures.

XCS conducts a number of random processes while performing covering and Darwinian genetic operations, two activities that belong to the discovery component. In covering, generalization allows every created rule contains one or more wildcards, determined by a probability variable. Darwinian mutation and crossover assist the original system in attempting to seek better rules from the good ones, but not without payoff. Randomizations in those two techniques cause the system making "uneducated guesses" in the process of generalizing a classifier. It slows down the learning rate and potentially costs a number of correcting requirements. Despite the fact that many investigations have attempted to address this issue, yet no satisfying breakthrough is introduced.

RC goes to the same direction with Darwinian genetic operation does. It considers each classifier as an individual and focusing on the condition attribute, which is in form of array of information. But, instead of performing mutation or crossover, the elements of the array are manipulated using a technique, based on the induction concept. So, each rule in the population is considered as an observation datum while each offspring becomes a conclusion of the learned knowledge. From the heap of individuals, the proposed system attempts to find any underlying pattern, leading to one or more possible conclusions.

Unlike the implementation of Darwinian genetic operation, every step in RC is operated heuristically, following a specific algorithm. Generalized rules express conclusions, which are taken based on the proposed system's experience. Referring to Darwin's quote presented at the beginning of this chapter, the conclusion is made after a series of observations. Proper generalizations in a learning system can only be performed by having a sufficiently comprehensive data. Such accurate basis makes the conclusion have a greater chance of being a correct one. XCS-RC handles the issue concerning rule accuracy very thoroughly starting from covering process, as described in the following subsection.

3.1.1 Covering

One of the main differences between XCS and XCS-RC is that the proposed system does not implement generalization in covering (see Figure 3.2). Every time an input is received, the system checks any existence of matching rules in the population. In case covering is required, one or more classifiers are created using the exact details of the input without wildcards. This is an effective way to prevent any occasional misplaced wildcards. Later, when any of those new rules is picked to enter the action set, the environmental feedback is used to update the classifier for the first time. This way, the accuracy of each rule is kept high.

Such technique gives a significant impact to the system. At some moment while performing the learning cycles, an induction process occurs. Here, each experienced rule is regarded as a datum. A high amount of accurate data means that the collected knowledge is considerably comprehensive, leading to a greater chance of making adequate conclusions. In case one or more inaccurate data exist, the potential of making inadequate conclusions appears. Therefore, XCS-RC maintain the accuracy of each learning result by not performing any generalization attempts in covering. In practice, the wildcard probability parameter $P_{\#}$ is permanently set at zero.

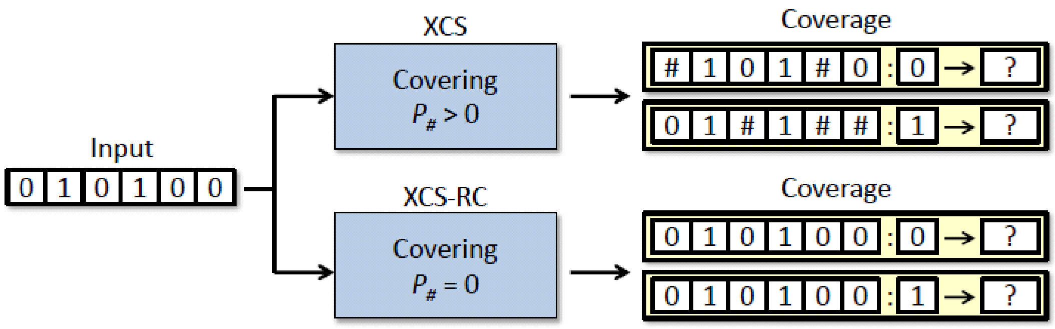


Figure 3.2: Comparison of covering implementation

Eliminating generalization in covering ensures the accuracy of the newly created rules. When an inexperienced classifier enters the action set and then gets updated, it becomes an experienced and ungeneralized rule with high accuracy. The removal of generalization eliminates the difficulty of determining a proper value for $P_{\#}$, as well as the necessity for an evaluation of this probability variable. Furthermore, it also reduces the possibility of having one or several misplaced wildcards, which frequently causes a slow learning rate.

All experienced and specific (not general) rules are confirmed accurate. Commonly, since those individuals do not contain any wildcards, the population grows quite quickly. Such situation can be disadvantageous. But at the other hand, a high number of accurate individuals may also express a set of comprehensive knowledge, which is very useful for making conclusions. The following subsection discusses this matter in more details.

3.1.2 Generalization

In scientific activities, making conclusion can only be conducted after analyzing some sets of data. Therefore, before performing induction, XCS-RC attempts to find any underlying patterns within the classifier's condition. This means, that each classifier in $[P]$ is regarded as an observation result, and some conclusions are about to be made based upon the available collection.

To perform generalization in an appropriate way, a user-defined parameter *combining period* (T_{comb}) regulates how often RC is performed. In other words, XCS-RC attempts to combine the rules in every T_{comb} learning cycles. Setting this parameter too high potentially causes the system to occupy more space than necessary, but an extremely low value may lead XCS-RC to repeatedly making inaccurate generalizations. To obtain the maximal advantage, the user should consider the characteristics and complexity of the problem, and properly determines a value for T_{comb}.

The system makes each conclusion following a specific pattern. Referring back to the example illustrating an unknown sequence in Figure 1.1, the whole individuals can theoretically be discovered after a valid conclusion is made. In XCS-RC, each rule is considered as a datum. Since $cl.C$ consists of several independent values, the pattern can only be figured out after examining each element. The conclusion is directly related to a generalization, and each pattern leads to a wildcard placement. A wildcard reflects an element which has no influence in determining the output, or in a word: insignificant.

An example is provided using the following illustration. This so-called *Working Man* scenario involves a man that goes to the office at weekdays. There are three available transportation modes to pick, i. e., a bicycle, a bus and a car. Among those, a car is the quickest in getting him to the workplace while the other two are comparable. The negative side of driving a car is the cost, which becomes one of the advantages of riding a bicycle. Using the mass transportation system costs less than driving the car, but still more compared to riding the bike.

In every morning, the man makes a decision after considering two variables representing the current situation, i. e., the weather and the remaining time before the official working hour. The former is expressed using two states (i. e., "RAINING" and "NOT RAINING"), while the latter is simplified into two situations, i. e., "SHORT" and "LONG". The goal is to get to the office on time without getting wet by the rain.

Such simple scenario can be transformed into a task which is suitable for classifier systems, as seen in Figure 3.3. After observing the environment, the man composes a binary input x and uses the available knowledge to pick an action a. Each every-day decision is either to go by a car, a bus or a bike, expressed by the codes 0, 1 and 2, respectively.

When the man arrives late to the office, a reward of 10 is assigned. Otherwise, being on-time or an early arrival is awarded by 50. Then, a reward 30 is given when he manages

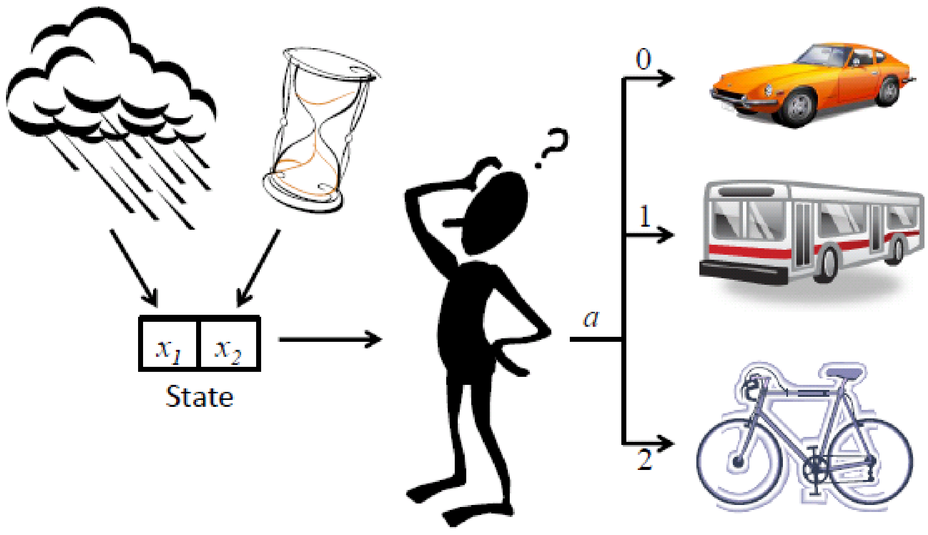


Figure 3.3: The working man scenario

Table 3.1: Working man's reward mechanism

No.	State	Payoff
1	Late arrival	10
2	On-time or early arrival	50
3	Got wet	10
4	Stayed dry	30
5	Maximum cost	10
6	Minimum cost	20
7	No cost	40

to stay dry or 10 for a set of wet clothes caused by the rain. Finally, riding a bike is the best option with respect to the cost. The man gives it a 40-point payoff, which is the highest compared to the bus with 20 and the car with 10.

By the time he arrives to the office, the man evaluates his latest decision following a reward mechanism given in Table 3.1. The values of the payoffs are weighed following his subjective opinion. After performing a number of learning cycles for some weeks, his collection of knowledge already covers all possible combinations of states and decisions.

Figure 3.4(a) presents the list containing the whole results of the activities. Each row represents a unique pair consisting of a specific situation and an option, followed by the man's quantified evaluation. A situation consists of a combination involving the weather and the time availability, while the option expresses the chosen transportation mode. E. g., the first row of the list can be translated into *"IF the weather is NOT RAINING, AND the time left is NOT MUCH, AND the chosen transportation mode is THE CAR, THEN the sum of the payoffs is 90"*. The highest payoff of each combination of states is illustrated with the green color.

The structure of information in the Working Man scenario is very similar to that employed by a common classifier system. A combination between the weather and the time availability represents the state kept in *cl.C* while a set of possible actions (i. e., car, bus and bicycle) are

No.	Raining?	Have Much Time?	Transp.	Evaluation
1			CAR	90
2	FALSE	FALSE	BUS	65
3			BIKE	60
4			CAR	90
5	FALSE	TRUE	BUS	105
6			BIKE	120
7			CAR	90
8	TRUE	FALSE	BUS	65
9			BIKE	60
10			CAR	90
11	TRUE	TRUE	BUS	105
12			BIKE	100

(a) Evaluation results

No.	Condition	Action	Pred.
1	00	0	90
2	00	1	65
3	00	2	60
4	01	0	90
5	01	1	105
6	01	2	120
7	10	0	90
8	10	1	65
9	10	2	60
10	11	0	90
11	11	1	105
12	11	2	100

(b) Classifier representations

Figure 3.4: Population for the Working Man scenario

stored in $cl.A$. And finally, $cl.P$ is determined by calculating the sum of the received payoffs, following the Table 3.1. In conclusion, the best option for the state "00" is driving a car with a reward at 90. This means, when the current situation is *"NOT RAINING and NOT MUCH TIME LEFT"*, then taking a car is the most desirable transportation mode. Figure 3.4(b) illustrates the population of rules representing the whole knowledge.

Up to this point, no generalized rules are involved in the population, meaning no wildcards are placed in any of the classifiers' condition. Induction can be implemented by performing RC to the population of knowledge. To give an impression on how each conclusion is made, the classifiers in $[P]$ are rearranged following three steps:

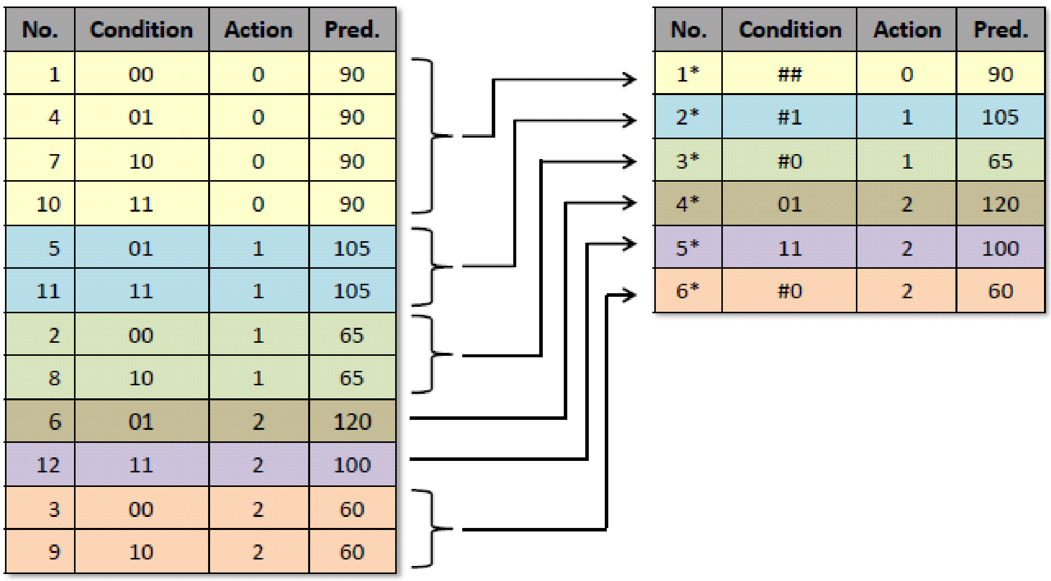
1. Grouping based on the action $cl.A$
2. Sorting based on the prediction $cl.P$
3. Creation of offspring by performing induction to the condition $cl.C$

Figure 3.5 visualizes those steps by illustrating the grouped and sorted classifiers along with the generalized ones. The varying cell colors denote the process involving the current rules and the corresponding offspring. RC creates the children (cl_*) following three regulations:

1. The number of bits owned by the child is equal to the parents
2. If the value of a bit position i is different, a wildcard is placed in $cl_*.C[i]$
3. If the value of a bit position i is identical, the value is inherited by $cl_*.C[i]$

The attributes of the generalized children are determined using such heuristics. Those regulations drive the production of cl_{1*} based on the conditions of four classifiers, i. e., cl_1,

cl_4, cl_7, and $cl_1$0. Since all of them have identical actions and predictions but their conditions are unique, $cl_{1*}.C$ is composed with two wildcards ("##").



No.	Condition	Action	Pred.
1	00	0	90
4	01	0	90
7	10	0	90
10	11	0	90
5	01	1	105
11	11	1	105
2	00	1	65
8	10	1	65
6	01	2	120
12	11	2	100
3	00	2	60
9	10	2	60

No.	Condition	Action	Pred.
1*	##	0	90
2*	#1	1	105
3*	#0	1	65
4*	01	2	120
5*	11	2	100
6*	#0	2	60

Figure 3.5: Generalization of rules for the working man scenario

RC successfully reduces the population size from 12 individuals to only six. Although the amount of classifiers is reduced by 50%, the existing knowledge is sufficient to cover all possible states without losing the accuracy. Referring back to the original story, such population can be translated into human's language, as expressed in Table 3.2.

Table 3.2: Translations of the generalized rules

No.	$cl.C$	$cl.A$	$cl.P$	Translation
1*	##	0	90	In ANY case, driving a CAR gets 90
2*	#1	1	105	If the time left is MUCH, taking a BUS gets 105
3*	#0	1	65	If the time left is NOT MUCH, using a bus gets 65
4*	01	2	120	If it is NOT RAINING, AND the time left MUCH, riding a BIKE gets 120
5*	11	2	100	If it is RAINING, AND the time left is MUCH, riding a BIKE gets 100
6*	#0	2	60	If the time left is NOT MUCH, riding a BIKE gets 60

This simple scenario shows that the induction concept is very useful to generalize classifiers. It illustrates how irrelevant information does not play any role in determining the final result. E. g., wildcards in $cl_{2*}.C$ and $cl_{3*}.C$ confirm that the only factor determining the reward for taking a bus is the time availability. Furthermore, taking a bicycle to work is the most undesirable option when not much time left, regardless of the weather. In conclusion, a proper induction enables the system to find specific patterns which become the basis for generalizing the available knowledge.

However, for facing problems with higher complexity, such generalization technique has a potential of hastiness. The term *hasty generalization* expresses a situation where the conclusion is taken without a set of sufficiently comprehensive data. The missing information might be either unlearned or disobeyed by the system. A hasty generalization commonly leads to an incorrect conclusion and potentially ends with an inaccurate child. More details regarding hastiness and the proposed way to solve this issue are discussed in the following subsection.

3.1.3 Hastiness

In the history of science, an instance of hastiness is provided by the story of the Tychonic system. Tycho Brahe founded the theory in his attempt to explain orbital movements of the planets, the moon and the sun. His theory was considered valid for hundreds of years, and was involved in a long debate in astronomy. Although finally the scientific rivalry was won by Galileo's heliocentrism, Brahe's work was considered as the most accurate astronomical calculation without any involvement of optical tools, and was considered as a major scientific breakthrough.

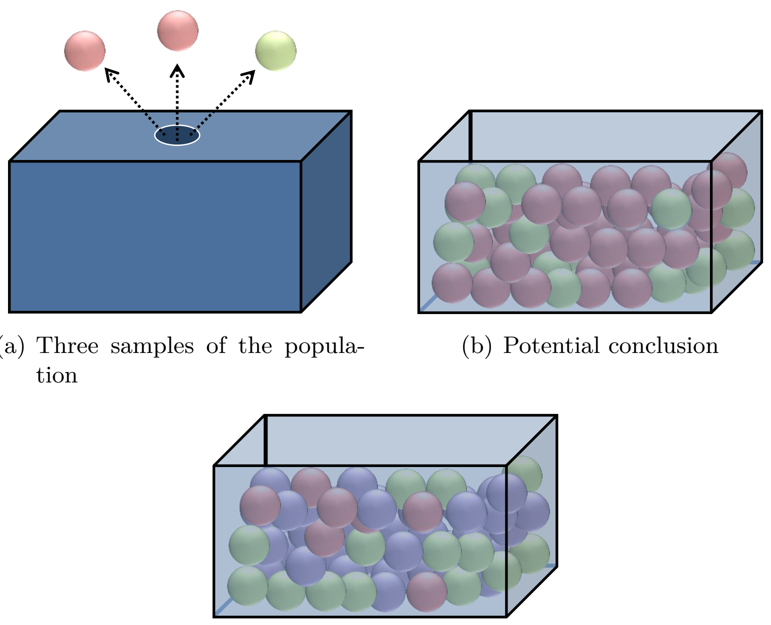
What did Brahe fail to recognize? In scientific debates against the Heliocentrists, the Tychonians argued that the earth is not orbiting since there was no indication that the stars had a change of positions. It can be concluded that the lack of supportive equipments can be considered as the cause, despite any possible pressures from the religious belief.

An evidence finally came in the $19th$ Century when Friedrich Bessel discovered a stellar parallax, a scientific fact that disproved the Tychonic system. Bessel conducted his experiments by using a number of quite sophisticated telescopes, which were not available in Brahe's time. This story provides a good explanation regarding the cause and the impact of hastiness in science. Not to claim that the Tychonians were careless for disobeying the scientific method, but mostly due to an absence of a relevant fact. Such hasty conclusion requires a correction in the future.

Another occurrence of hastiness can be illustrated by spinning the previously presented "black box" scenario. Figure 3.6(a) shows three individuals taken from a black box, where each is considered as a sample of the population. Such insufficient amount of data potentially misleads an observer to a conclusion depicted in Figure 3.6(b), which is significantly different from the actual condition (see Figure 3.6(c)). Unlike the Tychonic system, in this case, the hasty decision is made due to a lack of samples.

As a summary, a hasty conclusion is caused by two occurrences, i. e., non-comprehensive knowledge and insufficient number of samples. Those two might occur in XCS-RC, each with a potential of making incorrect generalizations. Therefore, in anticipating the first mistake that is illustrated by the Tychonian case, XCS-RC is equipped with a correction mechanism. After a generalized rule is executed and their attributes are updated, the proposed system monitors any indication of hastiness and performs a sudden act to avoid future confusions. A more detailed explanation regarding this matter is discussed in Section 3.4.

The second cause of hastiness should also be avoided. Therefore, the proposed system is designed to involve all available knowledge in the process of making conclusions. To increase the comprehensiveness of the learning result, a small modification on the action selection technique is applied, particularly when the exploration mode is used. The following section provides a discussion regarding this matter.


(a) Three samples of the population

(b) Potential conclusion

(c) Actual population

Figure 3.6: A hasty conclusion

3.2 Deterministic Explorative Decision

The chance of making a hasty conclusion can be reduced by implementing a number of techniques. Firstly, an anticipation is necessary. XCS-RC should possess a sufficiently comprehensive knowledge, which can only be obtained based on experience. However, the proposed system does not have the capability to demand or to determine a specific environmental state. So, it is designed to prioritize inexperienced rules, attempting to increase the amount of collected knowledge.

Figure 3.7 illustrates a simplified decision making mechanism, using the previously mentioned Working Man scenario. A population consisting of five experienced rules and an inexperienced classifier (denoted by the question mark at the *cl.P* column). Suppose XCS-RC runs a combining process on the existing $[P]$, the rules advocating the action "0" are combined and the child can be symbolized by "$\#\# : 0 \rightarrow 90$". Here, the system combines them by taking the risk coming from an unknown *cl.P* value owned by the inexperienced rule.

With another assumption that the RC process occurs later, an additional lesson can be obtained when XCS-RC receives an input "01". Such value can be matched by three classifiers entering $[M]$, where each of them advocates different *cl.A* values. Slightly afterwards, the action selection is executed to pick a winning output. Now, if the decision is taken using the exploitation mode, the winning action is "2".

A different situation may occur when the system currently uses explorative approach, which allows rules with a lower prediction value to become the winner. A common XCS mechanism suggests various techniques to pick a decision, e. g., roulette wheel or simple randomization. As an attempt of minimizing hastiness, XCS-RC uses a different decision making process. It deterministically picks a classifier that currently has the lowest experience, meaning that it intentionally chooses the action "0" in responding to the given situation.

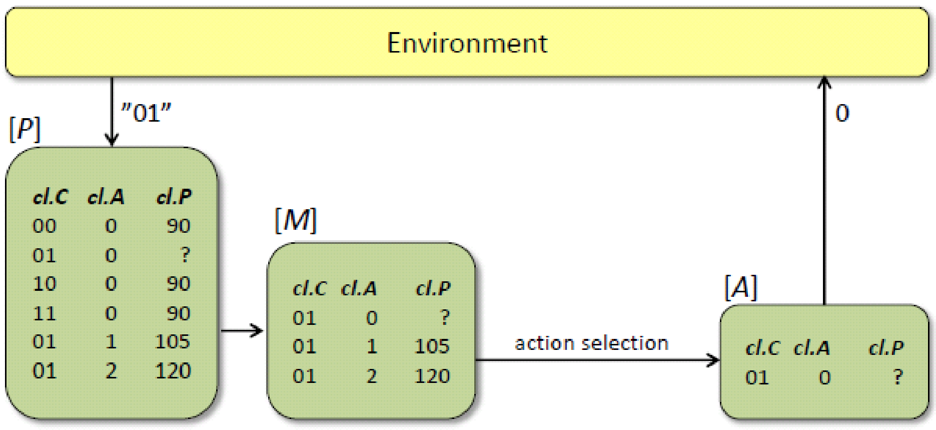


Figure 3.7: Determining explorative decision in action selection step

After the environmental feedback is received, the system obtains a new knowledge. The inexperienced classifier becomes experienced, and more useful for future combining activities. Using the scientific method as a reference, such deterministic explorative decision making is similar to conducting an experiment, attempting to discover any unknown facts. When an observer requires a set of specific experimental data, he picks some control variables to see the influence to the result. His data collection is expected to be more comprehensive after the experiment is conducted.

This technique can only be executed when the system receives a specific input that is covered by the inexperienced rule. Nevertheless, it contributes positively to the generalization result by providing a way to obtain more accurate data. The extra knowledge potentially leads to adequate combining processes and furthermore the right learning direction. In the following section, a full explanation of the RC implementation is presented, remarking the main idea proposed in this thesis.

3.3 Rule Combining

A classifier system employs reinforcement learning to collect its knowledge. Every time an environmental state is retrieved, the system attempts to behave appropriately by responding with a suitable action. Learning is performed based on the received reward at the next time step, denoting the suitability of the decision in responding to the previously given state. Along with that, the system has an effort to produce generalized classifiers without losing the accuracy.

Since XCS-RC does not perform generalization in covering, wildcard placements are solely determined by the experience. This technique guarantees that generalized classifiers have a high accuracy, although at the other hand, each of them can only cover a specific input. After the number of learning cycles reaches T_{comb}, the first RC mechanism is executed and the effort of producing general and accurate rules begins. Classifiers are picked pairwise, a pair at a time. Every pair of rules cl_i and cl_j should fulfill RC basic principles consisting of three criteria:

$C1$. The parents advocate the same action ($cl_i.A = cl_j.A$)

$C2$. The difference of the parents' predictions is tolerable ($|cl_i.P - cl_j.P| \leq predTol$)

*C*3. No disproving rules exist in the population

The previously described scenario involving a working man provides an example of a generalization process using the induction concept. However, the actual process is expected to have a higher difficulty level since the number of observed states in most scenarios is greater than two. Moreover, there are many scenarios that involve more actions, as well as a more complex environmental reward map. Hence, in order to generalize the classifiers in the population, XCS-RC performs a so-called *combining steps* procedure, which is explained as follows.

3.3.1 Combining Steps

Each generalization attempt is implemented by executing a series of combining steps. Figure 3.8 depicts a procedure performed by the proposed system, in the attempt of producing generalized classifiers. Starting from the left-hand side, XCS-RC collects rules having the same action and stores them into a so-called *combining set* (symbolized by [*C*]) in Step 0.

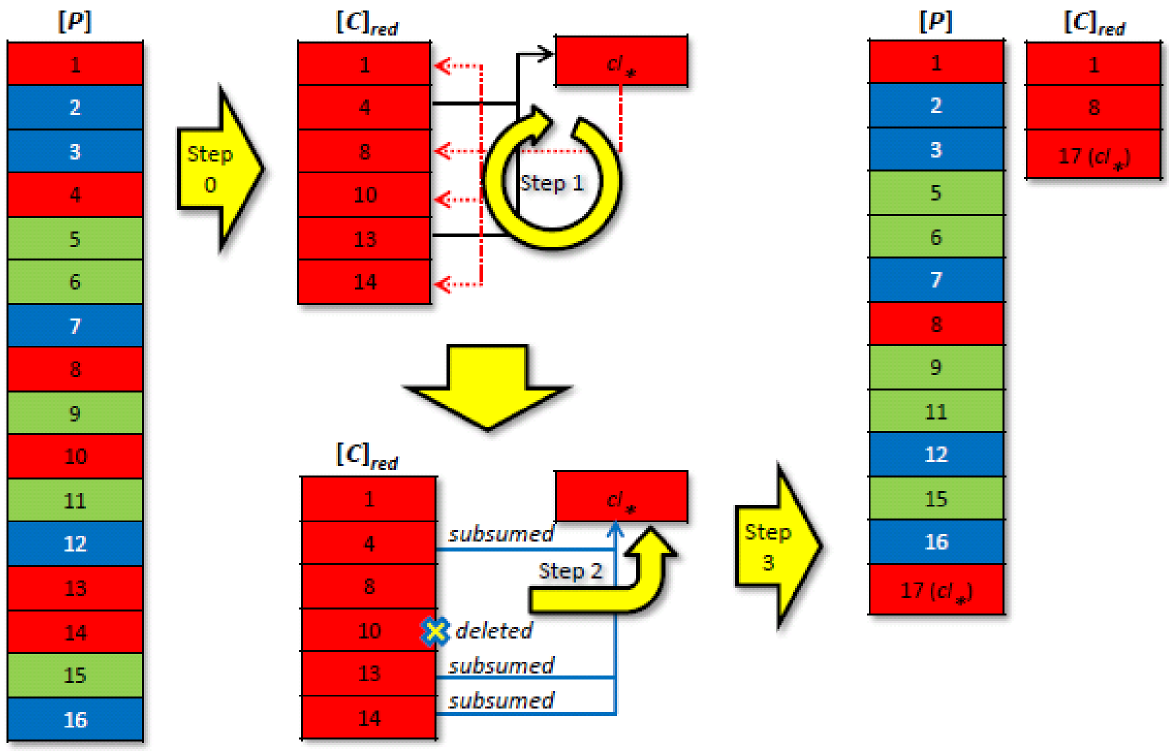
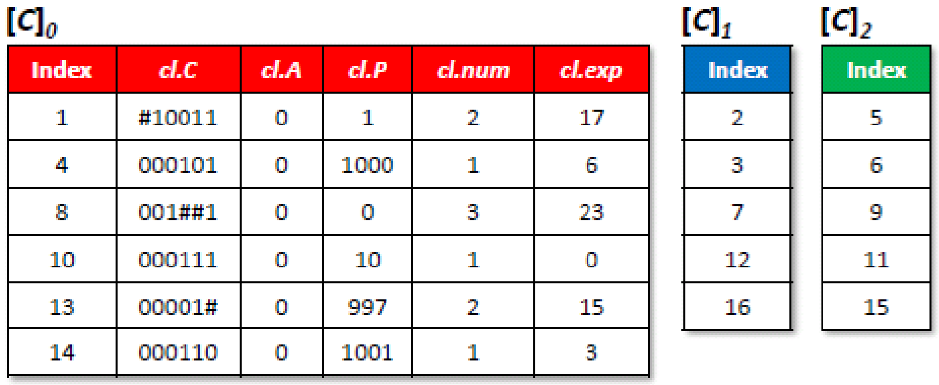


Figure 3.8: Overview of the combining steps

All generalization attempts are conducted in [*C*]. In Step 1, a parent-picking mechanism is performed, along with the attempts of validating the combining results. Step 2 has the task to build the child's attributes based on the information obtained from the involved rules. Finally, Step 3 ends the process by inserting the combining result into [*P*] and the corresponding [*C*]. A more detailed description regarding the full procedure is provided as follows.

Step 0. Preparatory

In Step 0, classifiers having $cl.A = a$ are recruited to a combining set $[C]_a$, as depicted in Figure 3.9. This step is useful to avoid the chance of violations to the criterion $C1$ of RC basic principles. Also, it emphasizes the fact that individuals with different action have no influence to the generalization in any way. All further processes are executed within each combining set $[C]_a$.



$[C]_0$

Index	cl.C	cl.A	cl.P	cl.num	cl.exp
1	#10011	0	1	2	17
4	000101	0	1000	1	6
8	001##1	0	0	3	23
10	000111	0	10	1	0
13	00001#	0	997	2	15
14	000110	0	1001	1	3

$[C]_1$

Index
2
3
7
12
16

$[C]_2$

Index
5
6
9
11
15

Figure 3.9: Combining sets

Each rule is copied into one and only one $[C]$. Commonly, the combining sets have varying amount of classifiers. One containing a larger number of individuals may require more time for performing RC. At the other hand, the chance of making a hasty conclusion tends to be lower. The process continues by performing parent picking processes, followed by combining and disproval examination.

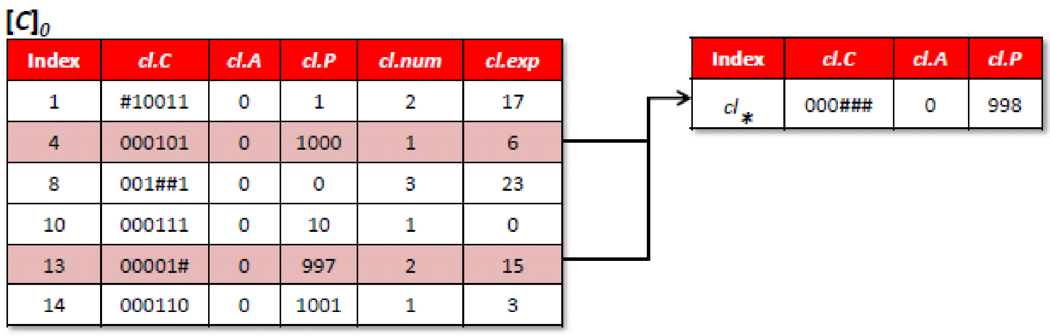
Step 1. Child candidate and its validation

XCS-RC forms a child candidate cl_*, one at a time. All pairing rules cl_i and cl_j should follow the criterion $C2$, i. e., having insignificant difference at their $cl.P$ values. This constraint distinguishes the quality of the action in responding to the covered state, while the significance level is determined by a threshold variable $predTol$ denoting a *prediction tolerance* parameter.

In determining which pair should be processed first, the system considers the resemblance level of the parents. This mechanism can be executed using two options, i. e., ascending and descending. The former states that two rules having the most similar $cl.C$ to be picked first, while the latter advises the contrary. Using ascending sequence can be more useful for handling scenarios with less input elements, since the number of wildcards tends to be low. Otherwise, a descending strategy is more favorable for learning the environments with higher generalization possibility. Considering that XCS-RC is designed to tackle various kinds of problems, the descending sequence is used as the default.

An offspring candidate inherits three main attributes from the parents (see Figure 3.10), i. e., the condition, the action and the prediction. While $cl.A$ is simply copied, $cl.P$ is determined by calculating the average value with respect to the classifiers' numerosity $cl.num$. Then, as an implementation of the induction concept, the value of $cl_*.C$ is derived from the parents following these rules:

- If positions p in the conditions of cl_i and cl_j are identical (i. e., if $cl_i.C[p] = cl_j.C[p]$), the value is copied to cl_* (i. e., $cl_*.C[p] := cl_i.C[p]$).



[C]$_0$

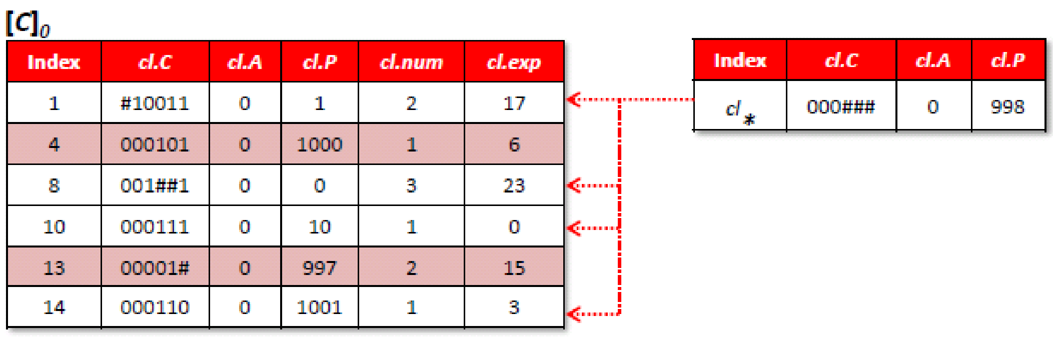
Index	cl.C	cl.A	cl.P	cl.num	cl.exp
1	#10011	0	1	2	17
4	000101	0	1000	1	6
8	001##1	0	0	3	23
10	000111	0	10	1	0
13	00001#	0	997	2	15
14	000110	0	1001	1	3

Index	cl.C	cl.A	cl.P
cl_*	000###	0	998

Figure 3.10: Forming cl_*

- If positions p in the conditions of cl_i and cl_j are different (i. e., if $cl_i.C[p] \neq cl_j.C[p]$), the value in cl_* becomes a wildcard (i. e., $cl_*.C[p] := \#$).

After forming three main attributes of the child candidate, the system runs an examination process to check any existence of a *disproving rule* in the corresponding $[C]$. As depicted in Figure 3.11, a conflict potential is detected by comparing cl_* to all experienced classifiers in the set.



[C]$_0$

Index	cl.C	cl.A	cl.P	cl.num	cl.exp
1	#10011	0	1	2	17
4	000101	0	1000	1	6
8	001##1	0	0	3	23
10	000111	0	10	1	0
13	00001#	0	997	2	15
14	000110	0	1001	1	3

Index	cl.C	cl.A	cl.P
cl_*	000###	0	998

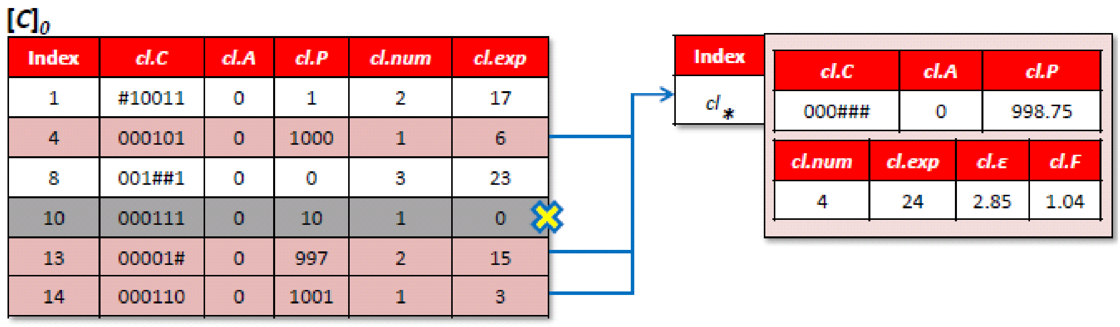
Figure 3.11: Examination of cl_*

An individual cl_k becomes a *disproval* for cl_* if they share a specific area within their input coverage, while their prediction values are far from equal. In other words, if both cl_* and cl_k are able to match at least one environmental state and their predictions are significantly different ($|cl_*.P - cl_k.P| > predTol$), then cl_* is considered a hasty conclusion.

The situation where cl_* is disproved leads the system to abandon it and to abort the current combining process. Afterwards, XCS-RC continues the generalization attempt by picking one of the remaining pair of parents. Otherwise, if no disproval is found, the process continues to equip the child candidate the necessary attributes.

Step 2. Attribute building

Step 2 starts by assigning a subsumer role to the child candidate cl_*. The subsumption technique in XCS-RC is a little bit different to that implemented in XCS. While basically both systems eliminate the subsumed individuals, XCS-RC takes the values of their attributes to build the ones belong to the subsumer. Figure 3.12 depicts an example of a subsumption process where the parents (cl_4 and cl_{13}) and another rule cl_{14} are subsumable to cl_*. An inexperienced classifier (e. g., cl_{10}) is simply deleted instead of being subsumed, meaning none of its attributes is considered for further processes.



$[C]_0$

Index	cl.C	cl.A	cl.P	cl.num	cl.exp
1	#10011	0	1	2	17
4	000101	0	1000	1	6
8	001##1	0	0	3	23
10	000111	0	10	1	0
13	00001#	0	997	2	15
14	000110	0	1001	1	3

Index
cl_*

cl.C	cl.A	cl.P
000###	0	998.75

cl.num	cl.exp	cl.ε	cl.F
4	24	2.85	1.04

Figure 3.12: Subsumption, deletion and calculation of cl_*'s attributes

In this step, $cl_*.P$, $cl_*.num$ and $cl_*.exp$ are calculated based on the subsumed classifiers. Firstly, the value of $cl_*.P$ is set following the average of the subsumed rules' predictions multiplied by their respective numerosity. In most cases, the difference between the new $cl_*.P$ and the initial value is insignificant. Then, the child's experience $cl_*.exp$ and numerosity $cl_*.num$ follow the sum of those owned by the subsumed rules.

Prediction error $cl_*.\varepsilon$ and fitness $cl_*.F$ are determined following a concept that the values should express the rule's accuracy. Although it is possible to use a more simple way (e. g., taking an average of the subsumed rule), a slightly more difficult calculation is composed. This choice is taken in order to give those attributes more "natural" values. Hence, the calculation of those two attributes refers to the process in $[A]$, where $cl.\varepsilon$ and $cl.F$ are updated every time the corresponding rule is executed. Equation 3.1 and 3.2 express the specific formula used in determining the values.

$$cl_*.\varepsilon = \begin{cases} \dfrac{|cl_*.P - P_I|}{cl_*.exp}, & \text{if } cl_*.exp \leq 1/\beta \\ \\ \dfrac{|cl_*.P - P_I|}{\lfloor 1/\beta \rfloor} \cdot (1-\beta)^{cl_*.exp - \lfloor 1/\beta \rfloor}, & \text{otherwise} \end{cases} \tag{3.1}$$

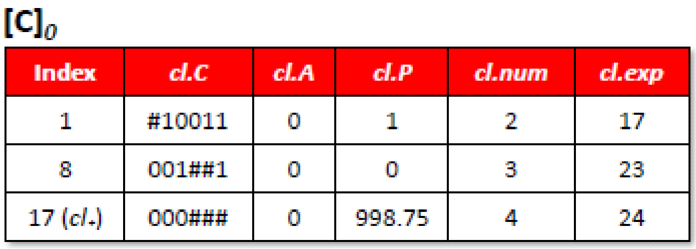
$$cl_*.F = (F_I - 1) \cdot (1-\beta)^{cl_*.exp} + 1 \tag{3.2}$$

Those equations have been derived experimentally, and their result resembles the outcome of an iterative process that starts with the default values for new classifiers (i. e., ε_I and F_I). In the iteration, $cl_*.\varepsilon$ and $cl_*.F$ are updated as if cl_* was the only member of an action set $[A] := \{cl_*\}$ and received a payoff $P = cl_*.P$ for $cl_*.exp$ times. These computations ensures that cl_* has proper values regarding the accuracy, with low prediction error and high fitness.

This way, the child has a similar characteristic to a "normal" classifier. Figure 3.12 depicts an example showing the new values of the offspring's attributes after subsuming cl_4, cl_{13}, and cl_{14}. Here, the parameter values are set $\beta = 0.2$, $P_I = 10.0$, and $F_I = 10.0$. The values of $cl_*.\varepsilon$ and $cl_*.F$ are 2.85 and 1.04, respectively.

Step 3. Insertion

XCS-RC inserts cl_* into $[P]$ and the corresponding $[C]_i$ in Step 3. At the same time, all subsumed and deleted classifiers are removed from both sets. Afterwards, the process goes back to Step 1 and the proposed system continues to execute RC procedures by picking one of the remaining pair of parents. The newly created rule can be involved as a parent, and also has a right to disprove a younger candidate.



$[C]_0$

Index	cl.C	cl.A	cl.P	cl.num	cl.exp
1	#10011	0	1	2	17
8	001##1	0	0	3	23
17 (cl_*)	000###	0	998.75	4	24

Figure 3.13: $[C]_0$ after insertion

Figure 3.13 depicts the rules in $[C]_0$ after cl_{*2} is accepted and becoming cl_6 (the previous indexing is kept for tracking purpose). Every time a child is created via combining, the number of rules in $[C]$ and $[P]$ is reduced. The updates of $[C]$ and $[P]$ denote the end of a generalization process, but not the whole RC mechanism. XCS-RC continues to check the remaining pairs, attempting to discover other patterns and use them for combining.

All generalized rules have the full capability like normal classifiers. They are allowed to enter $[M]$ and $[A]$, to become a parent in combining, and also to disprove a generalization process. Obviously, rules with wildcards have a greater chance of becoming a disproval. Here, one may figure that a disadvantageous situation can happen. An offspring that is created through an inadequate generalization potentially disproves several combining attempts, which are actually valid. Despite that this matter commonly occurs in environments with a large space input, the chance should be anticipated. Discussion regarding this matter is provided in Chapter 6 where XCS-RC is equipped with an additional function for identifying irrelevant features.

3.3.2 Algorithmic representation

In practice, XCS-RC attempts to perform generalization by executing a set of commands depicted in Algorithm 1. Firstly, the system composes a series of combining sets (line 1). Then, it searches a pair of combinable parents (line 6) and composes a candidate cl_* based on them (line 8). To prevent hastiness, a checking mechanism attempts to find any existence of disprovals (line 9). If at least one disproval is found, then cl_* is abandoned (line 11). Otherwise, the system equips the child with necessary attributes (line 13) and inserts it to $[P]$ and the corresponding $[C]$ (line 14). Finally, subsumed and deleted individuals are removed from both sets (line 15).

The creation of the child rises up a flag namely $ChangedMade$, which indicates that a change occurs in the population (line 16). Then, either the last combining attempt is successful or not, the system attempts to find other combinable parents from the remaining individuals (line 18). Finally, the system jumps back to line 4 if the $ChangeMade$ flag is up. Otherwise, if the last check in $[C]$ does not produce any offspring, the mechanism stops and the RC attempts ends.

Algorithm 1 Combining steps

```
1: COMPOSE a series of $[C]$
2: for all $[C]_i$ do
3:   initialize variable $ChangeMade \leftarrow$ TRUE
4:   while ChangeMade is TRUE do
5:     $ChangeMade \leftarrow$ FALSE
6:     FIND a combinable pair of rules
7:     while a combinable pair is found do
8:       COMPOSE a candidate rule $cl_*$
9:       CHECK for disprovals
10:      if disproval exists then
11:        ABANDON $cl_*$
12:      else
13:        BUILD attributes of $cl_*$
14:        INSERT $cl_*$ to $[P]$ and corresponding $[C]_i$
15:        REMOVE subsumed and deleted rules from $[P]$ and $[C]_i$
16:        $ChangeMade \leftarrow$ TRUE
17:      end if
18:      FIND next combinable pair of rules
19:    end while
20:  end while
21: end for
```

However, the disproval checking cannot guarantee that the offspring is not hasty. Although the result of the examination indicates an appropriate conclusion, it is possible that a valid disproval has not been learned. When this occurs, the knowledge represented by the child might contradict to a learning result in the future. The following section specifically discusses the anticipation to such issue, getting the system prepared to receive some "updated knowledge".

3.4 Knowledge Correction

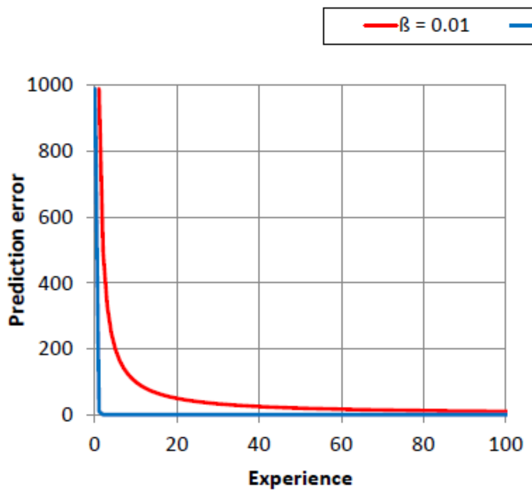
When making a conclusion, the system can only rely on data which are currently available. A number of the proposed children would be categorized hasty when they cannot pass the examination. Each of the successful ones makes a way to be accepted as full classifiers, symbolizing a conclusion of the subsumed individuals.

However, the chance that a conclusion is actually inappropriate is still open. Such hasty generalization occurs when the result is composed based on non-comprehensive data. In other words, the conclusion would have been disproved if the disproving knowledge is already learned. This is similar to the story of the Tychonic system, a theory that was known valid for centuries, and finally disproved by new scientific facts. Hence, beside the preventive mechanism using disproval examination, an additional procedure to anticipate a knowledge correction is necessary.

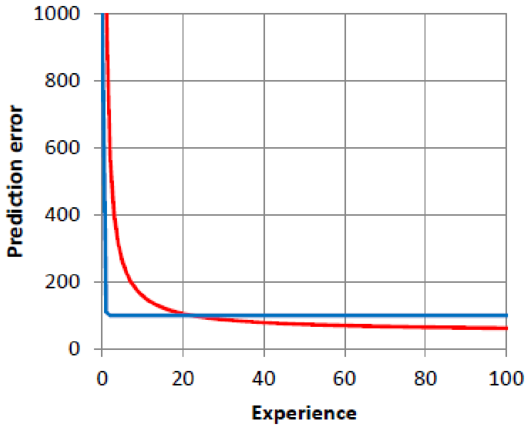
As explained in Chapter 2, prediction error and fitness are used to indicate a rule's accuracy. Both are updated within the action set, every time after the system receives a feedback from the environment. However, the purpose of those attributes are quite different. A classifier cl_i

is considered accurate if $cl_i.\varepsilon$ is lower than a predefined parameter ε_0, while $cl_i.F$ is used to denote the relative accuracy of cl_i compared to all the other classifiers in $[A]$.

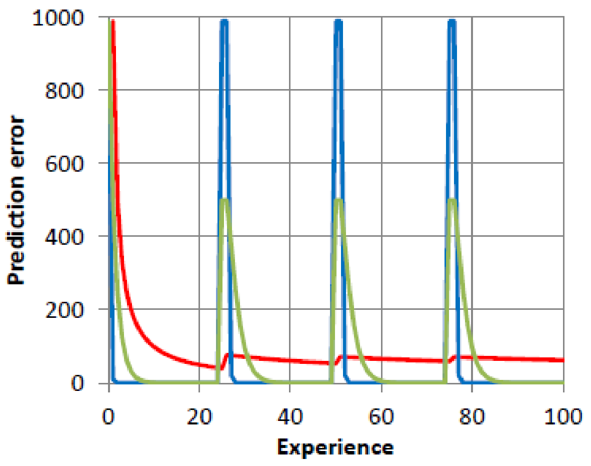
XCS-RC attempts to detect any indication of hastiness by monitoring the prediction error. The idea could be described this way: *"if a rule cl_i is accurate, then updating it shall not make the value of $cl_i.\varepsilon$ becomes higher"*. In conclusion, by monitoring the prediction error's value, an indication of inaccuracy can be detected and responded properly.



(a) Constant rewards at 1000



(b) Alternating rewards at either 950 or 1050



(c) Inconsistent reward for every 25*th* update

Figure 3.14: The values of $cl_i.\varepsilon$ over $cl.exp$

Figure 3.14 provides some graphical representations showing the value of $cl.\varepsilon$ over $cl.exp$ using various settings of the reward discount factor parameter β. The value of prediction error is observed by running a series of simulations based on the following assumptions:

1. No change of values in the parameter setting except a control variable β.

2. For each update, cl_i is a sole member of $[A]$ (no other rules are involved).

3. The feedback value is arranged in a way so that $cl_i.\varepsilon$ forms an observable pattern.

The first simulation involves a consistent reward value at 1000. As seen in Figure 3.14(a), every update tends to stabilize $cl_i.\varepsilon$. The blue line depicts the simulation result using $\beta = 0.99$, indicating a steep descent on the observed variable at the beginning of the simulation. Meanwhile, the red with $\beta = 0.01$ shows a more gentle behavior. From these facts, one can conclude that the value of prediction error never increases for both extremely high and extremely low reward discount factor.

A different payoff arrangement is given to the second simulation. Figure 3.14(b) depicts the prediction error value obtained by feeding XCS-RC with alternating feedbacks, i. e., either 950 or 1050. Unlike the previous simulation, the value is found stable at around 100 for the blue line and slowly getting lower for the red. In conclusion, the maximum value of $cl_i.\varepsilon$ tends to follow the difference of the alternating feedbacks, which is 100.

Those two instances indicate that the parameter β affects the calculations of prediction error. And for both of them, $cl_i.\varepsilon$ tends to be stable when the system receives a set of relatively consistent feedbacks along the run. This is the starting point of designing a knowledge correction mechanism, as Figure 3.14(c) depicts the result from an experiment that employs another scenario in order to get a more comprehensive conclusion.

The third simulation is arranged involving a significantly different reward at every $50th$ experience. Practically, the system receives 49 feedbacks at 1000 and then followed by a low reward at 0. The response is monitored to validate the idea that a hastiness control can be implemented by monitoring $cl_i.\varepsilon$. For a better impression, a third run using $\beta = 0.50$ joins the simulation, depicted by the green line.

All lines show their common behaviors for the first 49 iterations, which is as expected. The blue line depicting the result for $\beta = 0.99$ is the quickest to reach zero with only five iterations while the green requires 22. The red line ($\beta = 0.01$) only gets as low as 39.6 until the $49th$ update. However, the value of $cl.\varepsilon$ increases significantly after receiving a low reward. From previously zero, the blue line shows a steep increment to 990.00 while the green reaches 500.00. The value of the red line becomes 76.54 from previously 39.60.

Such significant change after receiving the low reward indicates that detecting hastiness is applicable by monitoring $cl.\varepsilon$. By taking the value from a prediction error tolerance parameter $predErrTol$, the proposed system employs a threshold for a flagging mechanism. Once the value of $cl_i.\varepsilon$ goes beyond the allowed limit after an update, cl_i is removed due to an indication of hastiness.

However, removing rules solely based on such threshold potentially arises another issue. The system might accidentally delete accurate classifiers with low experience, since commonly such rule has a high $cl.\varepsilon$ value. Hence, in the implementation, deletions can only occur to cl_i if these two conditions are fulfilled:

- The value of $cl_i.\varepsilon$ is beyond $predErrTol$
- The value of $cl_i.\varepsilon$ before the update is lower than afterwards

Applying these conditions prevents a situation where accurate rules are removed only because of having a low experience. The experiments conducted for composing this thesis show that this hastiness deletion mechanism does not have a tendency to delete good rules. Hence, this monitoring technique is employed as the part of the knowledge correction mechanism in XCS-RC.

Furthermore, such mechanism emphasizes the importance of determining a proper value for β. A value between 0.20 and 0.33 is quite common, including in the experiments conducted for composing this thesis. Nevertheless, some adjustments might be necessary, depending on the characteristics of the assigned task. Finally, the following section closes this chapter by summarizing a set of user-defined parameters employed by XCS-RC, in comparison to the original XCS.

3.5 Parameters

The proposed system is designed to cope with various problems having several levels of complexity. Therefore, a set of parameters is provided to allow system adjustments in facing varying tasks. Most of the original system's variables are still employed by XCS-RC. The difference mainly exists in the group "Discovery". Here, XCS-RC uses only four parameters instead of six. Along with the reduction, some variables in the category "Subsumption and Deletion" are slightly modified.

Table 3.3 presents a set of new system parameters employed by XCS-RC. Firstly, the variable *combining period* T_{comb} denotes the number of learning cycles has to be taken before the next RC attempt. Commonly, environments with higher complexity require more knowledge since having more classifiers tends to lead to a comprehensive conclusion. At the other hand, keeping the period short might speed up the RC process, since the system focuses on fewer rules.

Every time the generalization mechanism is executed, the system applies a filter to pick the parents. The aim is to guarantee that only a sufficiently experienced rule can become a parent. In practice, classifiers entering the action set less times than the parameter *minimum experience* $minExp$ cannot be considered as parents. For learning a simple scenario, assigning the value $minExp = 1$ is mostly sufficient. Obviously, it might occur that in facing a more complex task, a greater number could be more useful.

Table 3.3: New parameters in XCS-RC

Symbol	Name	Functionality
T_{comb}	Combining period	Number of trials between RC attempts
$minExp$	Minimal experience	Lowest $cl.exp$ value to become a parent
$predTol$	Prediction tolerance	Highest difference of $cl.P$ between parents
$predErrTol$	Prediction error tolerance	Hastiness detection threshold for $cl.\varepsilon$

After identifying the experienced rules, the proposed system copes with the available pairs of classifiers by combining them, one at a time. Following the basic principles of XCS-RC, the difference of $cl.P$ between the pairing rules must not be significantly different. Here, *predTol* serves the purpose by determining the allowed tolerance. A difference of predictions beyond that value prevents two individuals from becoming a pair of parents.

Finally, every time an update occurs to the rules in $[A]$, the new deletion mechanism is executed. A parameter *prediction error tolerance* $predErrTol$ determines a deletion threshold. This means, whenever the value of a $cl_i.\varepsilon$ exceeds $predErrTol$, cl_i is potentially removed. More detailed information regarding the removal is described in Section 3.4.

By employing these new parameters, the classifier system performs a discovery component following a specific heuristics named Rule Combining. The system is expected to behave more convincingly without any involvement of random processes. It is also important to emphasize the fact that analyzing the new parameters becomes more simple, and furthermore, adjusting their values to the user's need becomes less difficult.

Up to this point, the RC technique is proposed as a novel way to handle scenarios involving binary environments. Since complex environments are more likely to be represented with continuous values, the following chapter is intended to provide the discussion regarding real inputs. Such environment is less artificial, and the development to this direction is expected to increase the capability of the system to cope with some more complex problems. And more importantly, how induction can contribute to learn a real valued environment.

CHAPTER 4

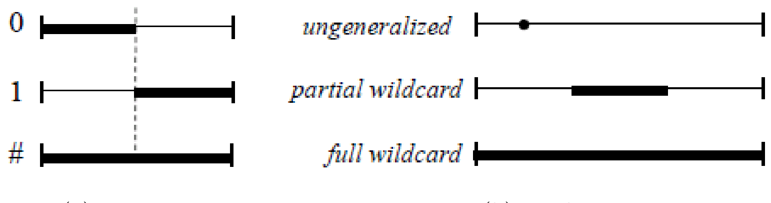
XCS-RC FOR CONTINUOUS VALUES

"And to make matters worse:
complexity sells better."

(Edsger Dijkstra)

Most problems in the real world that require machine learning techniques are the complex ones. At least, they are represented using various data types instead of only binaries. Some scenarios that are mentioned earlier (e. g., multiplexer) can be considered artificial, only used for experimental purposes. The capability of handling continuous input would make the proposed system one step closer to the aim of designing machines that assist humans. Therefore, instead of only handling binaries, XCS-RC is prepared to tackle more complex problems involving real values.

In [37], an extension of XCS with a capability of handling real value is introduced. The system is called XCSR (the letter "R" might stand for "Real"), and it adopts most of the concepts that are used in XCS. Obviously, the system introduces a major difference with respect to the classifier's condition. The whole performance and reinforcement components (i. e., matching, action selection, feedback and updating) are still implemented like the original. Also, the classifiers still advocate discrete outputs and their predictions still contain real values.



(a) Binary (b) Real

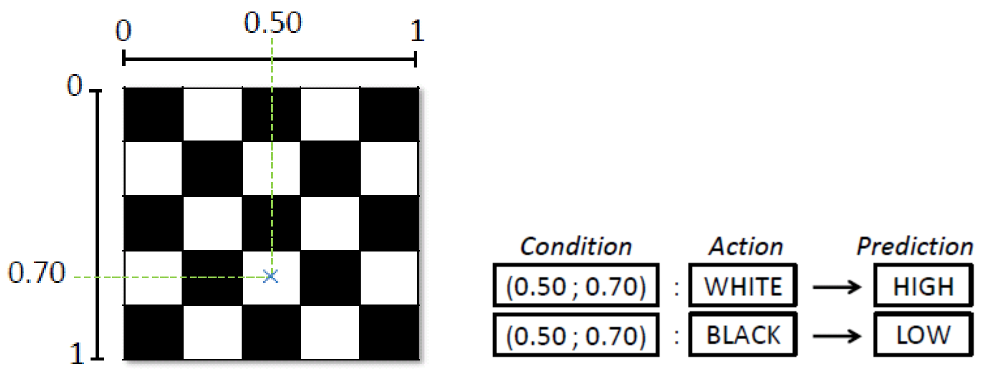
Figure 4.1: Types of coverage within an element of a classifier's condition

One of the most interesting discussions regarding the transformation from discrete to continuous values is defining the range of a wildcard. This matter should be addressed properly, since generalization is a key factor in determining the success of the learning process. For systems with binaries, a wildcard "#" is defined as a multi-valued entity with a capability of covering two specific values "0" and "1". In space representation, a wildcard has a full coverage of the range, from the lowest to the greatest possible input value (see Figure 4.1(a)).

At the other hand, continuous inputs can only be satisfied by classifiers containing interval representations, as seen in Figure 4.1(b). Here, the term "matching" is used when the current input is located within the covered area. Such interval represents a so-called *partial wildcard*, denoted by the limited range of input coverage. Then, another term namely *full wildcard* is introduced to express an interval that is capable of matching all possible input values. Rules without any intervals can only cover a single value (i. e., a dot), and it is categorized as a specific (non-general) one. In the following section, a discussion provides the practical use of such concept of input coverage.

4.1 Interval representations

Problems with continuous inputs are more difficult than the discrete ones. Not only due to a larger input space, but also in performing rule manipulations. Therefore, to equip classifiers with a capability of matching real inputs, the condition expresses its coverage using intervals. E. g., Figure 4.2 presents the illustration of a two-dimensional checkerboard problem.



(a) A dot (b) Rule coverage

Figure 4.2: A dot in a two-dimensional checkerboard

A classifier system learns such environment by receiving an input consisting of two elements, where each contains a value between 0.00 and 1.00. The pair of values works as a coordinate, representing horizontal and vertical position, respectively. As shown in Figure 4.2(a), a dot located at (0.50, 0.70) in a checkerboard is covered by a set of classifiers having a condition "0.50; 0.70" (see Figure 4.2(b)). A rule advocates the action "WHITE" while the other is "BLACK". A prediction "HIGH" is assigned to the one with an appropriate answer.

However, the conditions owned by those classifiers are only capable of covering a dot. Like a specific rule that has no wildcards in the binary case, having such ungeneralized rule would cost a large space for storage. Hence, the classifier system attempts to generalize such condition, placing wildcards in form of intervals, replacing those dots.

XCSR symbolizes an interval following a formula $int_i = (c_i, s_i)$, where c_i acts as the center and s_i denotes the half width of the spread (see Figure 4.3(a)). So, a classifier is able to match an input x if $c_i - s_i \leq x_i < c_i + s_i$ for every i. XCS-RC uses a different formulation to express intervals. Instead of having the center-spread relation, it employs pairs of lower-upper bound limits, as seen in Figure 4.3(b). In short, the coverage of a $[l_i; u_i]$ is equal to a $[c_i - s_i; c_i + s_i)$.

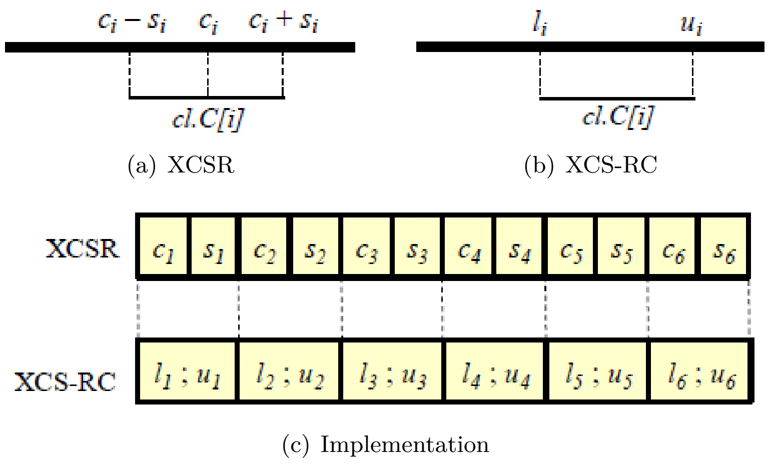

$c_i - s_i$
c_i
$c_i + s_i$
$cl.C[i]$
(a) XCSR
l_i
u_i
$cl.C[i]$
(b) XCS-RC
XCSR
c_1 s_1 c_2 s_2 c_3 s_3 c_4 s_4 c_5 s_5 c_6 s_6
XCS-RC
$l_1 ; u_1$ $l_2 ; u_2$ $l_3 ; u_3$ $l_4 ; u_4$ $l_5 ; u_5$ $l_6 ; u_6$
(c) Implementation

Figure 4.3: Intervals in XCSR [37] and XCS-RC [11]

Figure 4.4(a) presents an illustration involving an annotated region within a two-dimensional checkerboard. The area between the ranges [0.41;0.58] and [0.64;0.79] is recorded by XCSR using a set of pairing numbers {(0.50;0.09), (0.72;0.08)}, following the center-spread model (see Figure 4.4(b)). Different to that, XCS-RC comes up with a set of four values belonging to a pair that denotes the lower and upper bound of the intervals, as depicted in Figure 4.4(c).

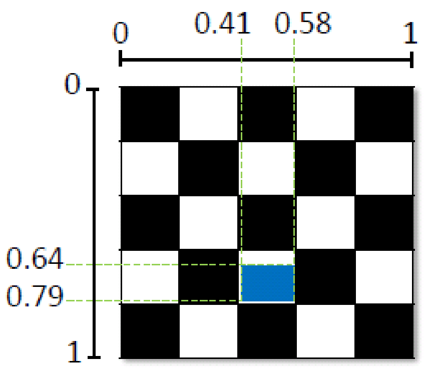
Both form of pairs are useful to express intervals, where each has its own advantages and disadvantages. The center-spread relation is more simple, denoted by having only one value within each element of the array. At the other hand, the lower-upper bound representation has a smaller number of elements, making it easier to cope with the inputs. Nonetheless, such difference influences the system quite significantly. The first effect is shown at the earliest process of the system, i. e., covering, which is discussed as follows.

4.2 Covering

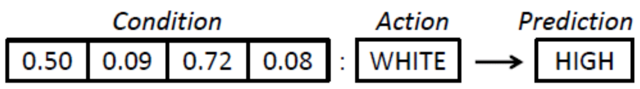
Different representation techniques between XCSR and XCS-RC comes from different rule manipulation methods, particularly concerning generalization. Here, XCSR still carries the previous concept from the original system, i. e., allowing generalizations in covering. As seen in Figure 4.5(a), every time covering creates a number of rules following an input x, the values of their conditions are determined using formulas $c_i = x_i$ and $s_i = rand(s_0)$. The latter means that the spread is determined using a random value between 0 and a user-defined parameter s_0.

Contrary to that, XCS-RC performs covering using the specifically received value, i. e., a dot. The mathematical representation for XCS-RC could be expressed using $l_i = x_i$ and $u_i = x_i$ for the lower and the upper bound, respectively (see Figure 4.5(b)). The result can be represented using the center-spread relation, by setting $c_i = x_i$ and $s_i = 0$.

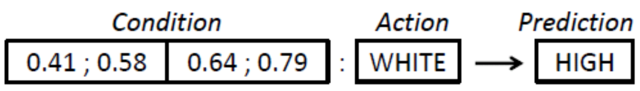
To emphasize the effect of having such different techniques, an example is provided using a simple rounding task. An input x with only one element contains a random value within a range [0.00;1.00]. The input is received by both XCSR and XCS-RC, where each of them is expected to pick the closest natural number as the action, i. e., either "0" or "1". In other words, when the input satisfies $x_i < 0.50$, the expected output is "0". Otherwise, the action "1"



(a) Coverage

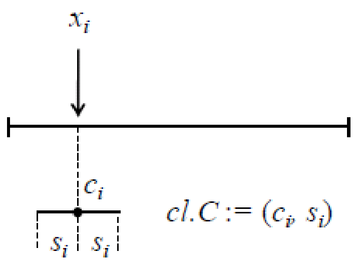


(b) XCSR

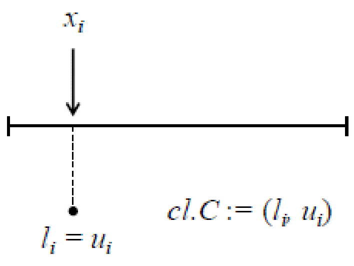


(c) XCS-RC

Figure 4.4: A 2-dimensional checkerboard with intervals



(a) XCSR



(b) XCS-RC

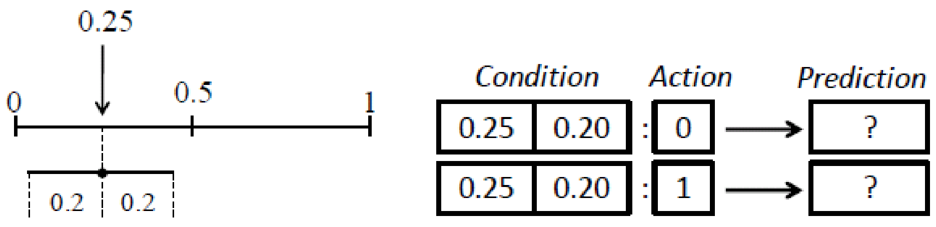
Figure 4.5: Covering

is more desirable. Each correct answer is awarded with a high reward, and a low one for a less expected output.

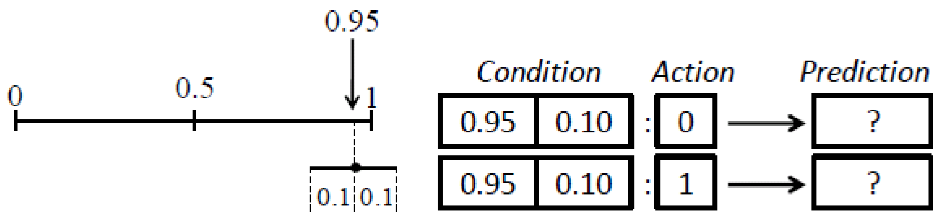
Every time an input is retrieved, XCSR seeks matching classifiers in the population. In case covering is necessary, one or more individuals are created. Figure 4.6(a) depicts an example where the received input $x_1 = 0.25$ is used to compose a new classifier. The value of the input is set to be the center of the interval, i. e., $c_1 = x_1$. To generalize the rule, a random process plays its role to determine the value of the spread, resulting $s_1 = 0.20$. From this example, a new classifier having an interval [0.05;0.45) is created.

A different situation appears by giving an input $x_1 = 0.95$, which is very close to the upper border. Any value of s_1 that satisfies $s_1 > 0.05$ leads XCSR to create rules that covers some unused area, as illustrated by Figure 4.6(b). Such situation does not affect the performance of XCSR in any way since the system is not aware of the existing border. So, an overlap outside the possible input range is tolerable.

Now, a disadvantage of performing generalization in covering is likely to occur when the value of the input is close to the rounding threshold. As seen in Figure 4.7, an input $x_1 = 0.40$ demands the output "0". However, if the spread is greater than 0.10, (e. g., $s_1 = 0.20$), the rule covers an overlapping area denoted by the red color. This leads to a potential mistake that



(a) Within the range



(b) Beyond the range

Figure 4.6: Covering for the rounding task in XCSR

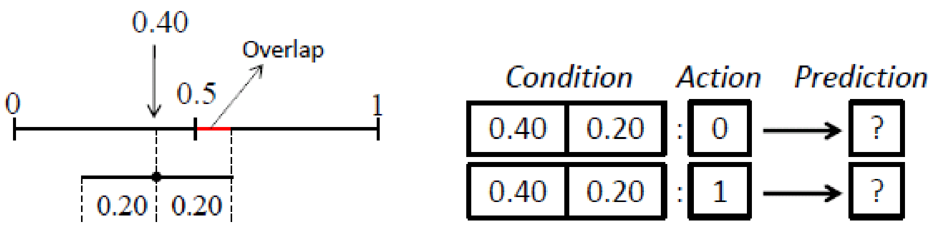


Figure 4.7: Covering overlap in XCSR

may occur in the future, e. g., when XCSR receives $x_1 = 0.55$. Such value will be responded by the system incorrectly with a "0" based on an inadequate generalization process in covering.

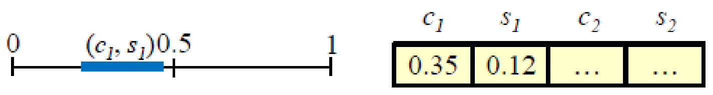
XCS-RC uses the same philosophy with its binary version to prioritize accuracy. The proposed system avoids placing any form of wildcards in covering, meaning no intervals are created while recording an unlearned state. Every input is covered using a dot to guarantee maximal accuracy. This way, when the first generalization attempt takes place, the population consists of a number of classifiers covering a dot. Therefore, a proper value for the minimal experience parameter $minExp = 1$ is strongly recommended when assigning the system to learn a real valued environment.

As mentioned previously, XCSR would perform the same covering technique to XCS-RC by setting $s_0 = 0.00$. However, such value is not advisable since the generalization methods owned by the learning systems come from different ideas. Induction is designed to handle rules having high accuracy, while genetic operation is more tolerant in that matter. More details on the implementation of Darwinian genetic operation in XCSR are described in the following section.

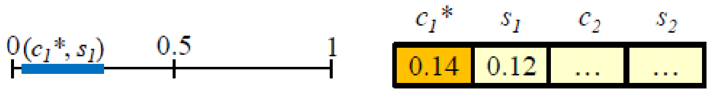
4.3 Darwinian Genetic Operators

Coverage overlaps that are created in covering might lower the rule's accuracy. In turn, they potentially affect XCSR's behavior in responding to inputs. When the system chooses an inadequate action, the learning rate slows down. Despite the chance that the technique may produce general rules with high accuracy, the negative impact should be handled properly. The Darwinian genetic operator is employed to do the assignment.

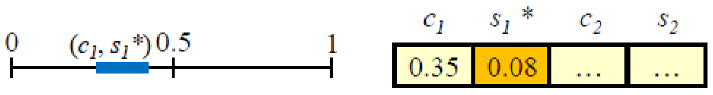
Like in the original XCS, XCSR attempts to tackle the issue by applying mutations and crossovers. The rules currently entering the action set $[A]$ are manipulated by setting them as parent classifiers to create some offsprings. The attempt is executed regularly, expecting that good rules will be composed this way. Basically, there is no difference between Darwinian genetic operations in XCS and XCSR. Both modify some elements of the conditions through mutations and initiate an exchange of a number of values in crossovers.



(a) Before mutating



(b) A mutation on c_1



(c) A mutation on s_1

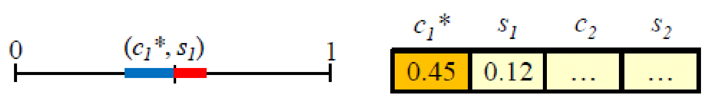
Figure 4.8: Mutation in XCSR

A mutation drives a change of value in one or more classifier's elements, leading to a modification of their input coverage. An instance for such process is provided by Figure 4.8(a), where a rule having $c_1 = 0.35$ and $s_1 = 0.12$ is picked as a parent. When the system performs a mutation to the classifier, a change of value might occur to the center, the spread, or both.

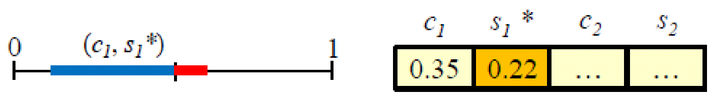
Figure 4.8(b) depicts a case when the center is modified. Here, the coverage of the classifier is shifted without changing the coverage width. Since the value of c_1 is reduced, the interval shifts closer to 0. Another case is depicted by Figure 4.8(c), where s_1 mutates and its value is becoming lower. As a result, the modified coverage is more narrow than the original. The third possibility of a mutation process is where both c_1 and s_1 are modified. This would cause a coverage shift along with a change in the interval's width.

Despite those changes are expected to overcome the issue regarding overlaps, unfortunately it also contains a drawback. When the value of s_1 is reduced, its coverage's width becomes more narrow. This may remove the overlap, but it also contains a loss of knowledge. The classifier becomes more specific, and the system might require additional learning cycles to recover the missing coverage.

Another negative factor of performing mutation is the potential of making overlapping intervals, which is a contradiction to the goal. Figure 4.9(a) depicts a change of c_1 that stretches the rule's coverage beyond the threshold. Similar to that, Figure 4.9(a) illustrates



(a) Mutation on c_1



(b) Mutation on s_1

Figure 4.9: Mutation and overlap

an overlap that appears after an increasing spread value. Performing a genetic mutation may cause such situation occur, leading to a slower learning rate. Unfortunately, crossover also has similar issues.

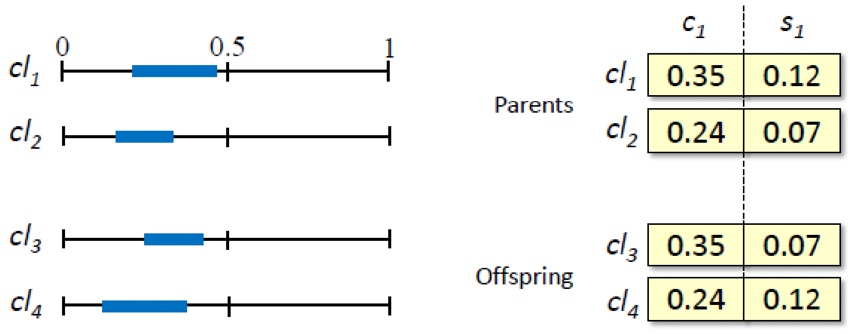
In performing a crossover operation, a pair of parents are selected and two offsprings are created based upon their values. As in the binary mode, an exchange of values is made between the parents. A one-point or two-point crossover remarks the elements that are being exchanged.

Figure 4.10(a) shows an illustration of a one-point crossover that is implemented to an array of two elements. A pair of rules cl_1 and cl_2 containing a respective center-spread (0.35; 0.12) and (0.24; 0.07) is chosen as parents. As seen in the illustration, two offsprings are created, i. e., cl_3 and cl_4. The former consists of a pair ($cl_1.C[1]$; $cl_2.C[2]$) while the latter has ($cl_2.C[1]$; $cl_1.C[2]$). Such coverages owned by the children are acceptable, since they contain no overlapping area.

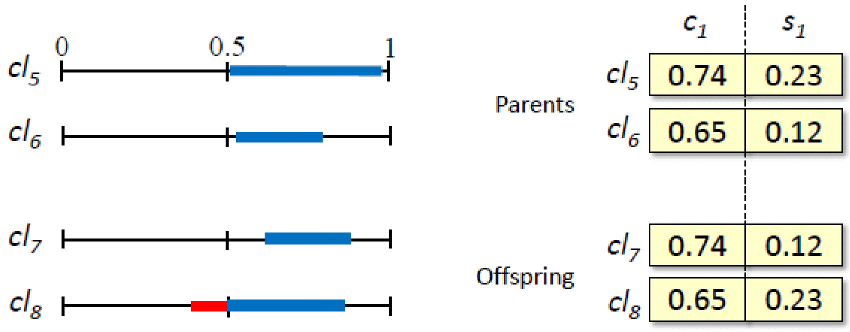
However, a mistake might occur as the case depicted by Figure 4.10(b). A pair of parents cl_5 and cl_6 having a respective center-spread (0.74; 0.23) and (0.65; 0.12) is involved in a crossover operation. The values of the center denote that the expected output for both parents is "1". Then, an exchange of value produces two new offsprings (i. e., cl_7 and cl_8) where each covers the range [0.62; 0.86) and [0.42; 0.88), respectively. By observing them, one can conclude that the latter contains an overlap at [0.42 ;0.50).

If cl_8 exists in the population, the system potentially fails to respond to an input within the overlapping range properly. E. g., an environmental state "0.45" that should be rounded to "0", will be categorized as a "1" by cl_8. Although the chance of providing the expected output still opens, the overlap obviously costs extra time consumption which makes the attempt of building a comprehensive knowledge inefficient.

Implementing Darwinian genetic operations have its own advantages and disadvantages. Since the technique are only applied to good rules, better offsprings are expected to enter the population afterwards. At the other hand, the previously mentioned case of overlapping intervals denotes the possibility of over-generality. In a response to that matter, the following section discusses how XCS-RC deals with real values without employing Darwinian genetic operation. Instead, it implements the concept of induction, similar to that introduced in the binary version.



(a) No overlapping area



(b) Overlapping area in cl_4

Figure 4.10: Crossover in XCSR

4.4 Combining

Over-general rules bring problematic situation to the classifier systems. It potentially causes XCS to respond to an input with an unexpected answer. At the very least, it slows down the learning rate. The proposed method offers a different philosophy of coverage modification. And as in the binary version, XCS-RC handles real value without relying on randomization.

Since XCS-RC operates ungeneralized covering that only produces dots, earlier combining processes normally cope with individuals with such specific coverages. Figure 4.11 depicts an illustration of a child candidate cl_* which is created by combining a pair of rules. Each of the parents covers a dot in the jth element of the condition. In general, three cases may occur in combining a pair of dots whose each element contains the values p_j and q_j:

- if $p_j = q_j$, then $cl_*.C[j] = p_j$
- if $p_j < q_j$, then $cl_*.C[j] = (p_j;\ q_j)$
- if $p_j > q_j$, then $cl_*.C[j] = (q_j;\ p_j)$

The first possibility happens very rarely and the formula is self-explanatory. Then, the second and the third ones are depicted in Figure 4.11(a) and 4.11(b), respectively. The value of p_j comes from $cl_1.C[j]$ while q_j follows $cl_2.C[j]$. The lowest value is picked as the lower bound of the interval, while the other determines the upper limit. This way, when cl_* successfully passes the examination, $cl_*.C[j]$ covers an interval expressed with $[p_j;\ q_j]$.

The explanation continues by focusing on the case shown in Figure 4.11(a). When the child enters the population, the symbolization changes from cl_* to cl_3. In the future, cl_3 has

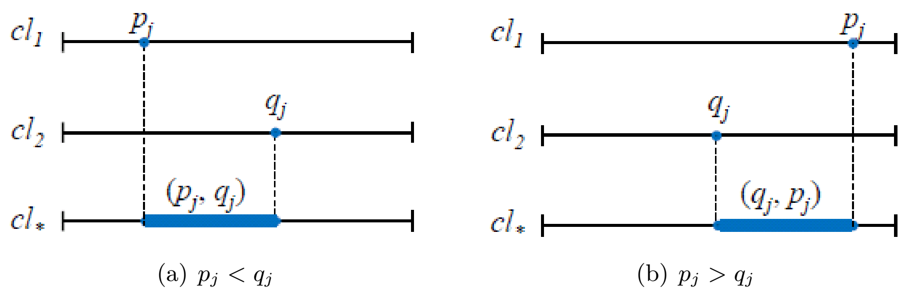


Figure 4.11: Combining dots in XCS-RC

an equal right to become a parent and a disproval as the others. The combining process that involves intervals can be categorized into several models, i. e., subset, overlapping and gapped.

Firstly, Figure 4.12(a) illustrates the situation where $cl_4.C[j]$ is a subset of that owned by cl_3. In such case, the value of $cl_*.C[j]$ is simply copied from $cl_3.C[j]$. The second possibility involves a pair of overlapping coverages, as depicted in Figure 4.12(b). Different from the previous example, cl_* inherits its lower bound from cl_5 while the upper is determined by that owned by $cl_3.C[j]$.

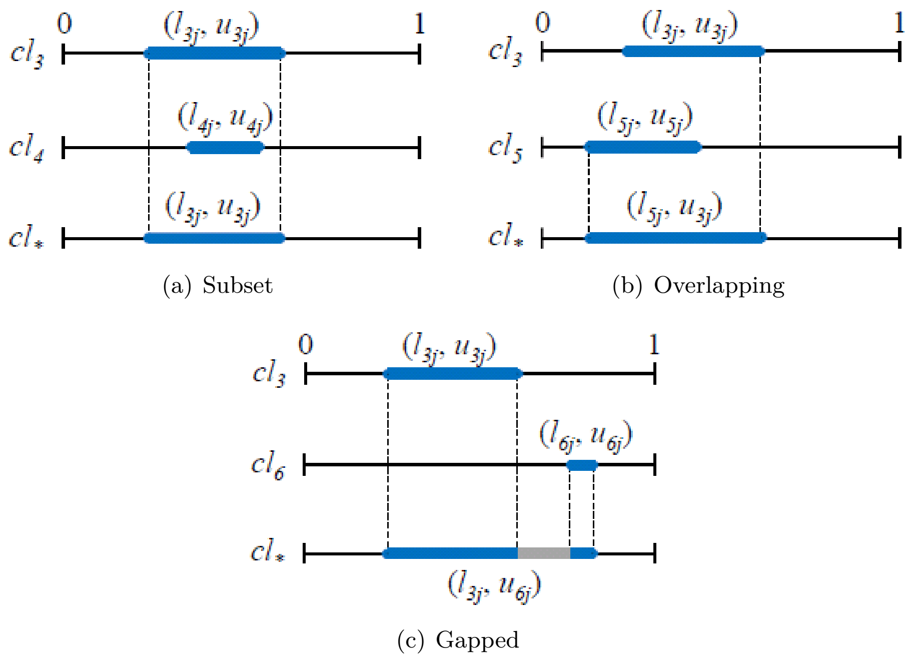
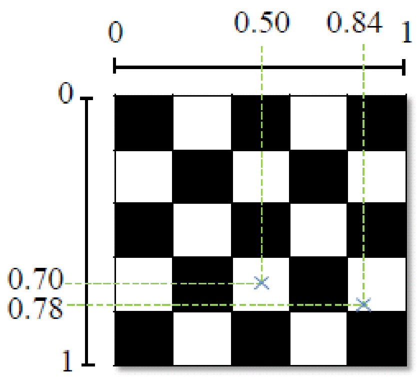
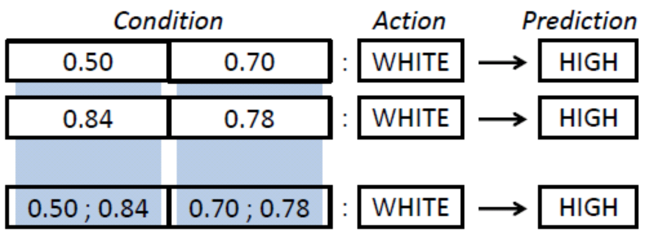


Figure 4.12: Combining intervals in XCS-RC

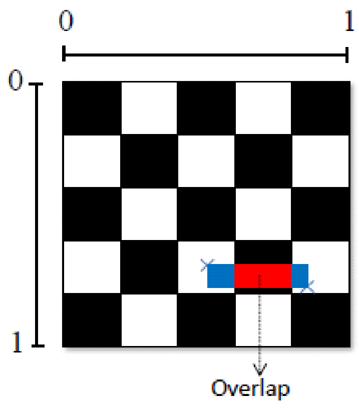
The third illustration shown in Figure 4.12(c) involves two intervals without any shared area. Similar to the situation a pair of dots is combined, a gap lays between the covered areas. Such occurrence arises the potential of performing an undesirable hasty generalization, and the combining attempt is potentially disproved by an experienced rule. This situation should be emphasized as a potential cause of having a low learning rate. To provide a better impression in this matter, an illustration involving the checkerboard scenario is presented as follows.



(a) Dots in the different white area of the checkerboard



(b) A pair of parent classifiers and the child



(c) The child's coverage

Figure 4.13: Combining drawback in XCS-RC for real value

Figure 4.13(a) depicts two dots located at (0.50, 0.70) and (0.84, 0.78), respectively. Assuming that no disproval exists, a pair of parents covering those dots are combined and the child covers a rectangular area shown in Figure 4.13(b). XCS-RC represents the offspring with a classifier "$[0.50; 0.84][0.70; 0.78] : WHITE \rightarrow HIGH$", as depicted in Figure 4.13(c). Later afterwards, when an input points a dot within the covered black area (e. g., at the coordinate (0.65, 0.75)), the system would respond to it incorrectly. XCS-RC attempts to prevent such failure using a similar technique to the one employed in the binary mode, i. e., disproval checking. A deeper discussion regarding the implemented modifications from the binary version of XCS-RC is provided in the following section.

4.5 Modifications from Binary XCS-RC

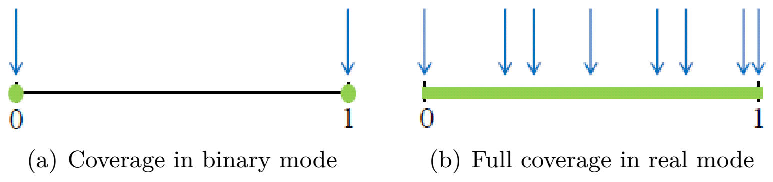
In real world, there are more problems expressed using real values than binaries. Therefore, one can say that a learning system that only capable of handling discrete values is too artificial. Fortunately in this case, most of the concepts used in designing RC are applicable to be implemented in handling real values. This means, with some technical adjustments, XCS-RC for binaries is capable of performing induction to rules that employs intervals.

The purpose of this section is to emphasize the key factors relating XCS-RC for binary and real values. Among three components employed by the system, the performance and the reinforcement parts require no modifications at all. So, the focus of the adjustment is in the discovery component, which prepares the proposed system to cope with a higher complexity. The discussion starts by describing the fundamental modification that is implemented to the proposed method, i. e., the concept of wildcard.

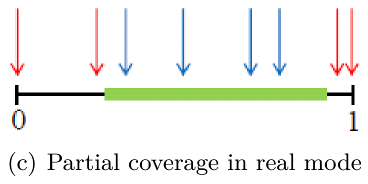
4.5.1 Wildcard

Both XCS-RC for binaries and real values generalize rules and place wildcards in the offspring based on the learned knowledge. The child enters the population after a disproval checking mechanism and attribute building. In both binary and real-value problems, a wildcard represents the capability of a classifier to cover a larger input space. Obviously, a difference characteristic of the input makes it being expressed differently.

Figure 4.14(a) illustrates the space representation of a wildcard in a specific bit position p of a condition $cl.C$. In facing a binary environment, the input has only two possible values (i. e., "0" and "1"), denoted by the black arrows. So, a binary wildcard simply covers those points without involving any values in between. Here, each coverage point is depicted by a blue node.



(a) Coverage in binary mode (b) Full coverage in real mode



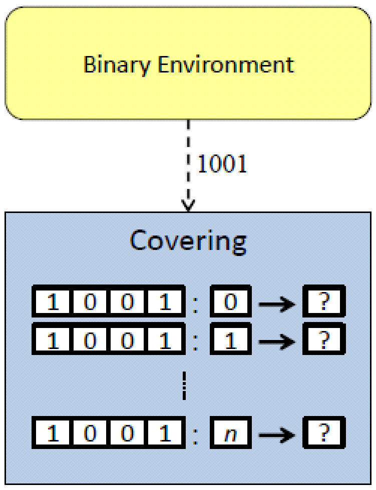
(c) Partial coverage in real mode

Figure 4.14: A wildcard at $cl.C[p]$ in space representation

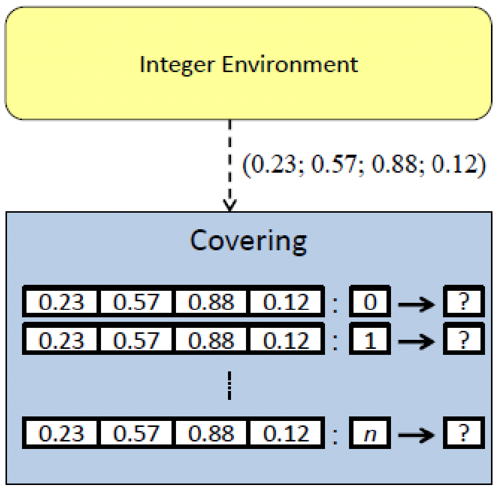
A different approach is used for handling real values, where wildcards are designed in form of intervals. As illustrated by Figure 4.14(b), a full wildcard has a capability to match any possible inputs (represented by black arrows). Then, another term *partial wildcard* is used to express an interval that does not cover the whole area. Such coverage is illustrated by Figure 4.14(c), where a wildcard is unable to match some states outside the coverage, depicted by red arrows.

4.5.2 Covering

XCS-RC is designed to start the learning process with an empty population. Hence, covering is the very first step it executes. The proposed system creates a set of rules to cover incoming unlearned states. Then, it assigns those classifiers with varying actions. In the implementation, those rules are created using an exact value of the input, denoting no generalization applied in covering.



(a) Binary



(b) Real

Figure 4.15: Covering on XCS-RC having n possible actions

Figure 4.15(a) and 4.15(b) depict a moment where XCS-RC learns different kinds of environment. For each type of inputs, a set of n rules is created, where n denotes the number of possible actions. All produced classifiers have neither wildcards ("#") nor intervals, denoting their non-generality. The disadvantage of maintaining such specific rules is the obvious requirement of a greater space, while the advantage is maximal accuracy. Since induction requires a set of accurate data to come up with adequate conclusions, covering without generalization is considered suitable to be implemented to XCS-RC.

4.5.3 Rule Combining

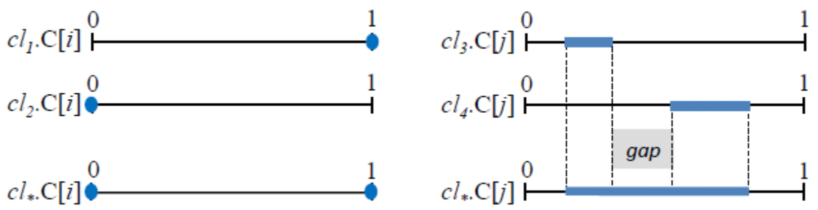
The RC's basic principles state that the parents must advise the same action, they have similar predictions and the population contains no disproval for the child. None of those guidelines requires any modifications for extending the capability of RC to cope with real values. Therefore, the difference is only determined by the techniques in manipulating the condition's coverage.

A conclusion is always made based on the available knowledge. In XCS-RC, an offspring is created after obtaining the values of some attributes owned by the parents. Figure 4.16(a) illustrates the space representation for combining in binary mode, where $cl_1.C[i]$ = "1" is combined with $cl_2.C[i]$ = "0". This process produces a child candidate cl_* having a wildcard in its ith bit. In space representation, the wildcard is depicted with two nodes where each of them covers "0" and "1", respectively (see Figure 4.14(a)).

Figure 4.16(b) depicts an example of a typical situation in a real-valued environment. Two intervals contained by $cl_3.C[j]$ and $cl_4.C[j]$ are combined into $cl_*.C[j]$. The lowest value of the covered area becomes the lower bound while the highest determines the upper limit. Different to combining two binary values, a coverage gap commonly exists in real mode. The binary version does not face similar issue since an input bit always contains either "0" or "1".

The system runs a typical disproval checking mechanism after creating cl_*. For both binary and real-value modes, an individual becomes a disproving rule cl_d when it fulfills these criteria:

$C1$. Both cl_* and cl_d advocate the same action

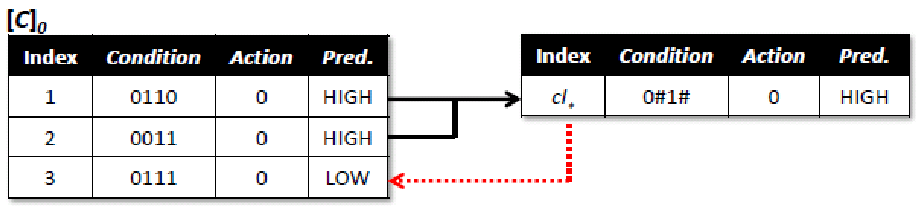


(a) Binary inputs (b) Real inputs

Figure 4.16: Combining in space representations

$C2$. Both cl_* and cl_d share an area of input coverage

$C3$. The difference of their predictions is beyond the tolerance parameter $predTol$

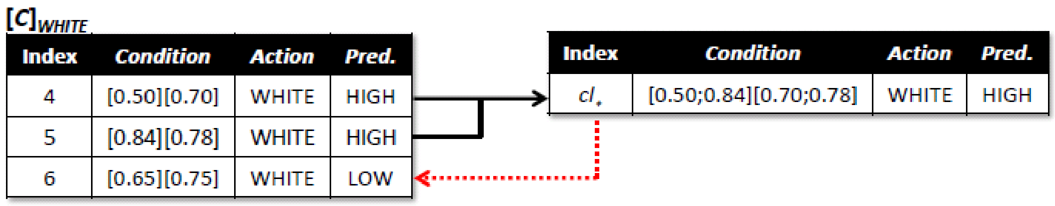


[C]₀

Index	Condition	Action	Pred.
1	0110	0	HIGH
2	0011	0	HIGH
3	0111	0	LOW

Index	Condition	Action	Pred.
cl_*	0#1#	0	HIGH

(a) Binary inputs



[C]$_{WHITE}$

Index	Condition	Action	Pred.
4	[0.50][0.70]	WHITE	HIGH
5	[0.84][0.78]	WHITE	HIGH
6	[0.65][0.75]	WHITE	LOW

Index	Condition	Action	Pred.
cl_*	[0.50;0.84][0.70;0.78]	WHITE	HIGH

(b) Real inputs

Figure 4.17: Examples of disproving classifiers

Figure 4.17(a) illustrates a combining process involving a pair of parents, cl_1 and cl_2. The candidate is created and a disproval examination is executed. Here, the system finds that cl_3 fulfills all three criteria of becoming a disproving rule. Similar situation is presented by Figure 4.17(b) where cl_6 disproves the combining result between cl_4 and cl_5. The coverage of cl_* (i. e., [0.50;0.84][0.70;0.78]) satisfies a dot in a coordinate (0.65, 0.75) which is pointed by cl_6. In both cases, considering that cl_* and the disproval have the same action but significantly different prediction, the child candidate is abandoned.

Those examples denote that the concept of disproving rule can be implemented in a similar way for facing both binary and real environments. However, the latter commonly has a greater chance of hastiness due to the large space of input. Moreover, combining real values often involves a gap, particularly at the earlier learning cycles. Setting a proper value for combining period T_{comb} might minimize the chance of hastiness. Thus, by giving XCS-RC a reasonable time for collecting necessary experience, the conclusions are expected to be more comprehensive.

Finally, in the following chapter, a series of comparisons is provided to validate the effectiveness of implementing induction. The experiments involve the original binary XCS, XCSR for real values and XCS-RC for both binary and real. To make a fair comparison, they are assigned to learn the same set of scenarios following some works from the previously published investigations.

CHAPTER 5 EXPERIMENTS

"In our play, we reveal what kind of people we are."

(Ovid)

After a series of descriptions concerning the proposed method, this chapter is solely intended for a validation. The judgement is made by assigning identical problems to XCS, XCSR and XCS-RC. The results are provided using a series of graphical representations in order to assist the readers in obtaining a better impression. Before discussing the decisive aspects, a short discussion is presented here as a short summary of the previous chapters.

The aim of implementing induction is to improve the system's effectiveness in learning the environment, as well as to increase its efficiency in discovering desirable classifiers. New rules are expected to be accurate and general, but not over-general. The original XCS has a success story in handling such challenge, but not very satisfying in terms of speed. Several evaluations on that matter were made, mainly concentrating on the Darwinian genetic operators. In conclusion, the involvement of several random variables slows down the learning process. Such randomization technique could be regarded as a way to produce "uneducated guesses", which is very difficult to evaluate. Therefore, a heuristic-based approach is introduced in this thesis, eliminating all random processes in the attempts of rule discovery.

XCS-RC does not make a conclusion by solely relying on the information obtained from a pair of parents. It is also designed to consider the remaining knowledge that might influence the result. Such technique minimizes the chance of producing a child from a hasty generalization. Along with that, a set of additional mechanisms and modifications are added to the proposed system, i. e., specific covering without generalization, deterministic action selection and hastiness detection. The whole package is expected to fulfill the purpose of improving the learning performance.

In conducting the experiments, identical procedures and parameters are applied to both systems to conduct a fair comparison. Three scenarios are assigned to the learning systems, i. e., multiplexer, checkerboard and Markovian environments. After describing each of those tasks, some comparisons are provided to emphasize the effects of the modifications. Finally, an evaluation is presented at the closing section of this chapter, describing some interesting issues on the XCS-RC's performance.

5.1 Procedure

XCS performs an online reinforcement learning, i. e., seeing each mistake as a lesson and broadening the chance to pick good responses for future decision makings. In the implementation, the system maps the environment by observing its states and then attempting to provide the best response possible. After a feedback is awarded, XCS updates the knowledge accordingly. A low reward following an unsuitable action is useful to acknowledge a less desirable decision, while a high one increases the probability of the output to be chosen again in the future when an identical state appears.

The experiments are conducted by running an XCS source code provided in [5], which is also the basis for developing the XCS-RC implementation. The modifications are mainly made for the purpose of discovering general and accurate rules, along with some other smaller improvements as well. Here, all lines that belong to Darwinian genetic operators and other unused methods are removed from XCS-RC's source code. Hence, one can be firm that the proposed method is executed by solely taking advantage of its own algorithm, with neither positive nor negative interference coming from the original method.

Table 5.1: Parameter setting for the experiments

Parameter	Value	Applies to
N	400	
α	0.1	
β	0.2	
γ	0.71	
δ	0.1	
nu	5.0	both
ε_0	0.01	
θ_{del}	20	
$teletransportation$	50	
p_I	10.0	
ε_I	0.0	
F_I	10.0	
θ_{GA}	25	
pX	0.8	
pM	0.04	
$P_{\#}$	0.33	XCS
$predictionErrorReduction$	0.25	
$fitnessReduction$	0.1	
θ_{sub}	20	
T_{comb}	100	
$predTol$	10	XCS-RC
$minExp$	1	
$predErrTol$	100	

All learning systems face identical scenarios throughout the experiments. Since the comparison is focused on RC's contribution to the learning process, known tasks from some previous publications are chosen for evaluation purpose. An identical parameter setting is given to all systems (see Table 5.1), except when stated otherwise while explaining a corresponding case.

5.2 Results and Comparisons

A series of experiments are conducted to observe XCS-RC's behavior. Meanwhile, for the original XCS, the results are obtained by executing the provided source code. Specifically for XCSR, the presented values are taken from a set of collected data from the original publication in [37], without any additional reruns.

Each judgement is determined by two main aspects, i. e., the performance (indicated by the difference to the expected value) and the population size (the number of (macro-)classifiers in $[P]$). The former should be optimized either maximally or minimally depending on the problem, while the latter should always be minimized. Commonly, each problem has a composition denoting an optimal population (symbolized by $[P]_{opt}$), which is the final aim of the proposed system. When all learning systems successfully reach an optimal value, a smaller number of cycles indicates a better capability of learning the environment. Each comparison involves a set of figures to provide a better impression on the results.

In composing this thesis, the systems are compared using three types of problem, i. e., binary input with single-step problem, binary/multi-step and real/single-step. The experiments for the single step problem employ multiplexer and checkerboard scenarios. Meanwhile, the multi-step problem is represented by some Markovian environments namely Woods and Maze.

5.2.1 Multiplexer with binary inputs

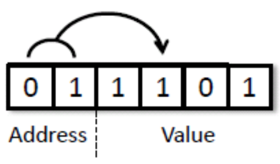
Scenarios involving multiplexer functionality are easy to understand. Binary (or sometimes called Boolean) multiplexer cases are used as the validation scenario in Wilson's original work [35] and the follow-ups (e. g., in [36]). A version involving the continuous valued input is also presented in [37]. The task is suitable for investigating the basic capability of a classifier system in learning single-step problems. Despite its simplicity, the case still offers sufficient challenge for a classifier system in performing a proper generalization.

In a common multiplexer scenario, the environment feeds a series of states to the learning system. All inputs are formed in array of l elements, where $l = k + 2^k$. The first k parts of the input point a specific position in the remaining 2^k, in which the expected answer is stored. The target for the learning system is to always pick the correct action as the output, thereby performing a 100% correctness rate.

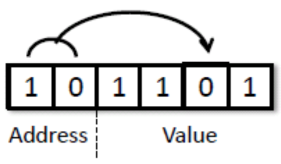
The conducted investigations use several values of $k = 2, 3, 4$. Each of them is named following the length of the inputs, i. e., the 6-bit, 11-bit and 20-bit multiplexer (will be mentioned as MP6, MP11 and MP20 afterwards). Figure 5.1(a) illustrates an MP6 input "011101", where the first k elements contain an address that points out a specific bit with a value "1". Some changes on the address bits would switch the answer into "0", as depicted by Figure 5.1(b). Such regulation is also valid for MP11 with $k = 3$, as shown by Figure 5.1(c) and 5.1(d).

Each learning system has two assignments. The first one is to get a stable 100% correctness rate, with some additional credits for the one reaching such expectation with a smaller number of learning cycles. The second assignment is performing a proper generalization. This is to emphasize the importance of reducing the number of required rules in responding to any incoming inputs.

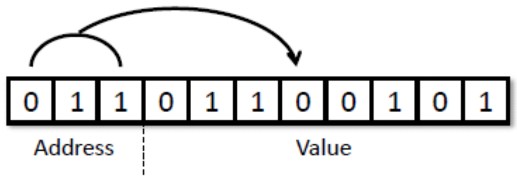
For MP6, $[P]_{opt}$ that provides the aims with respect to the performance and the population size is given by the set of rules depicted in Table 5.2. Each of the first eight classifiers has



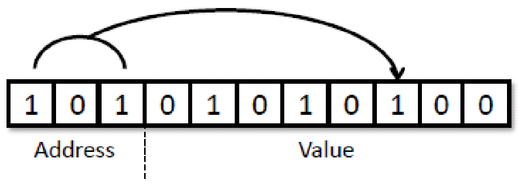
(a) An MP6 input with the answer "1"



(b) An MP6 input with the answer "0"



(c) An MP11 input with the answer "0"



(d) An MP11 input with the answer "1"

Figure 5.1: Examples of multiplexer inputs

a good reward prediction (i. e., 1000), while the rest advise the incorrect answer and having $cl.P = 0$.

In terms of quality, those 16 individuals are categorized as good classifiers based on two factors, i. e., maximal generality and accuracy. Both are indicated by the impossibility to add one or more wildcards without making it over-general. In performing an MP6 operation, a classifier system having $[P] = [P]_{opt}$ would achieve an obvious 100% correctness rate.

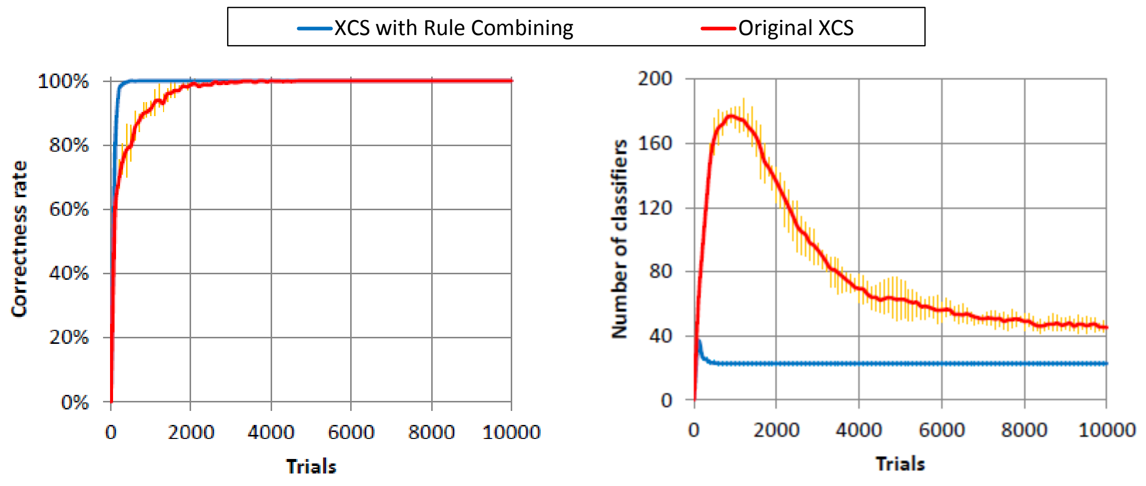
Table 5.2: $[P]_{opt}$ for the MP6 task

No.	Classifier	No.	Classifier
1	000###:0 → 1000	9	000###:1 → 0
2	001###:1 → 1000	10	001###:0 → 0
3	01#0##:0 → 1000	11	01#0##:1 → 0
4	01#1##:1 → 1000	12	01#1##:0 → 0
5	10##0#:0 → 1000	13	10##0#:1 → 0
6	10##1#:1 → 1000	14	10##1#:0 → 0
7	11###0:0 → 1000	15	11###0:1 → 0
8	11###1:1 → 1000	16	11###1:0 → 0

Each individual listed in Table 5.2 owns three wildcards, and therefore capable of matching eight different inputs. Such coverage is determined by 2^n, where n represents the number of wildcards. Using similar calculations, each rule in $[P]_{opt}$ for MP11 and MP20 is capable of covering $2^7 = 128$ and $2^{15} = 32\,768$ states, respectively. By comparing those coverages to the number of possible inputs, one can conclude that $[P]_{opt}$ for MP11 consists of 32 individuals while the size for MP20 is 64.

Every learning process begins after receiving the first binary input. Afterwards, it picks an action and prepares to receive the feedback. A reward "1000" is provided for every correct output, and zero otherwise. Both systems operate in alternating explore and exploit mode. Since this observation focuses on the system capability, only the values obtained from the exploit trials are recorded and then presented as the result of the experiments.

All provided graphics are obtained from an average of 20 runs, where each simulation is independent to the others. The depicted data for MP6 and MP11 are based on a sliding window for the last 50 exploit trials, while MP20 employs a width of 250 exploit trials. Additional error bars in the figures indicate the 25*th* and 75*th* percentiles. In some cases, the errors are very small, causing the bars to be not easily seen.



(a) Performance (b) Population size

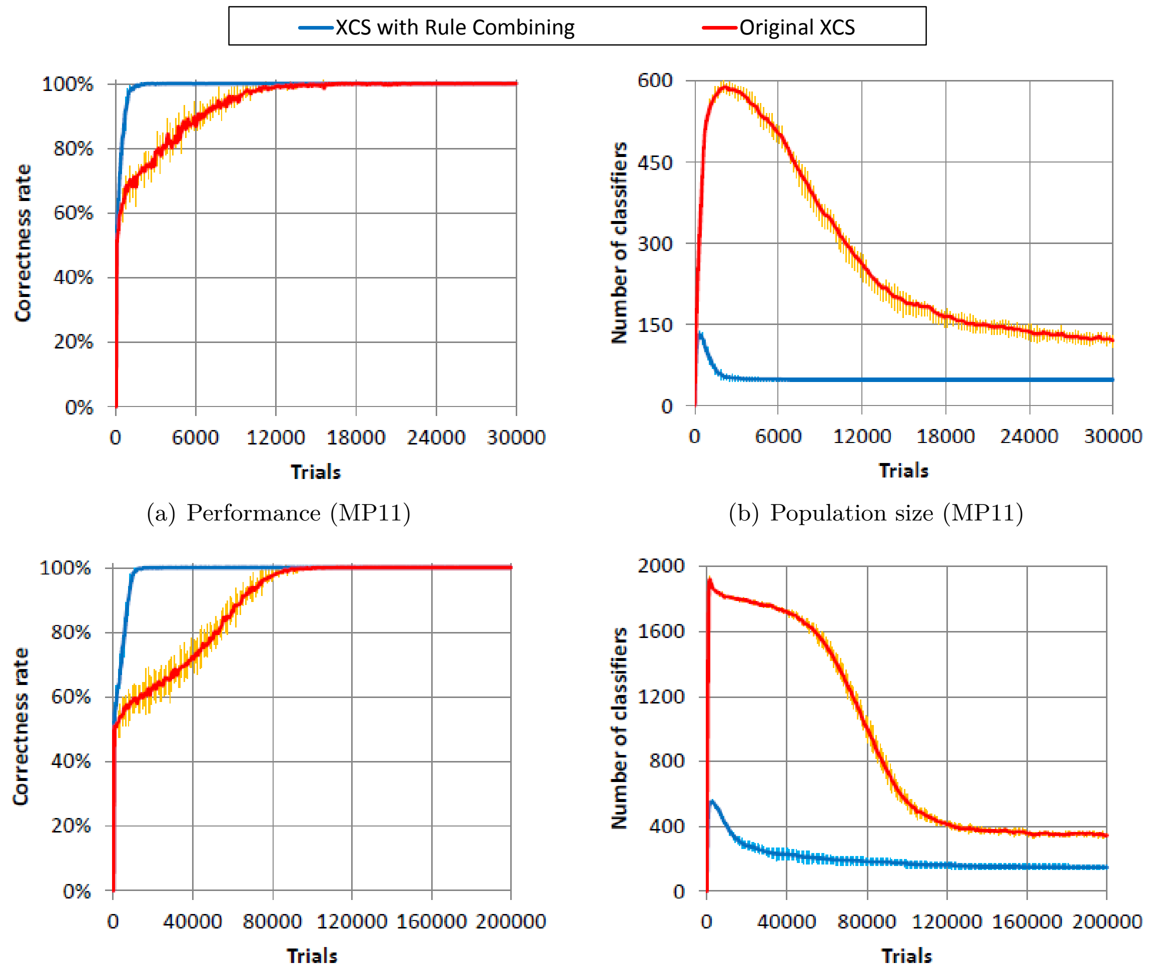
Figure 5.2: Results for binary MP6

Firstly, Figure 5.2 depicts two comparisons of the correctness rate and the population size between XCS and XCS-RC for MP6. As shown in Figure 5.2(a), the better achievement is performed by XCS-RC, which is represented by the blue line. The proposed system provides a full correctness rate more quickly than XCS (the red line). XCS-RC classifies all inputs correctly after approximately 700 trials (i. e., 350 explore and 350 exploit), while XCS requires more than 1000 trials to reach a stable correctness rate at 100%.

For the simulations employing the binary MP6, the setting for the maximum population size is $N = 400$ for both systems. Here, XCS-RC quickly reduces the number of individuals without reaching beyond 50. Around 400 trials are required to obtain a population size with less than 24 classifiers, which is sufficiently close to the minimum at 16. In comparison, the population of XCS is still converging after 10 000 trials, reaching a size of approximately 45 classifiers.

Similar situations are obtained from MP11 simulations (see Fig. 5.3(a)). Using a set of identical parameters to MP6 except $N = 800$, XCS-RC requires about 7000 trials to reach a stability performance at the full correctness rate. At the other hand, XCS still makes small mistakes after 29 000 trials. The comparison regarding the size reduction shows another superiority of the proposed method. The population of rules owned by XCS has not finished converging by the end of the simulations. After 30 000 trials, XCS's $[P]$ still contains around

120 classifiers. In contrast, XCS-RC maintains only 50 rules or lower since the 3700*th* trial.



(a) Performance (MP11) (b) Population size (MP11)

(c) Performance (MP20) (d) Population size (MP20)

Figure 5.3: Results for binary MP11 and MP20

Figure 5.3(c) and 5.3(d) present the results from the MP20 task using $N = 2000$ for XCS and $N = 800$ for XCS-RC. With such disadvantage, the proposed system is still superior by only requiring 15 000 trials to achieve a high correctness rate at 99.8%. It is significantly better than XCS, which needs more than 100 000 trials to perform at the same level. With respect to the population size, XCS-RC keeps at most 200 classifiers after 61 000 trials. In contrast, XCS never manages to reduce the size of $[P]$ under 340 classifiers until the end of the simulations at 200 000 trials.

The results of the experiments using three multiplexer cases lead to a temporary conclusion that XCS-RC is capable of solving classification tasks. Furthermore, the proposed method shows a more superior performance compared to the original version. For the binary multiplexer scenarios, XCS-RC reaches full correctness rate faster than XCS, and even quicker in reducing the number of classifiers.

5.2.2 Multiplexer with real-valued inputs

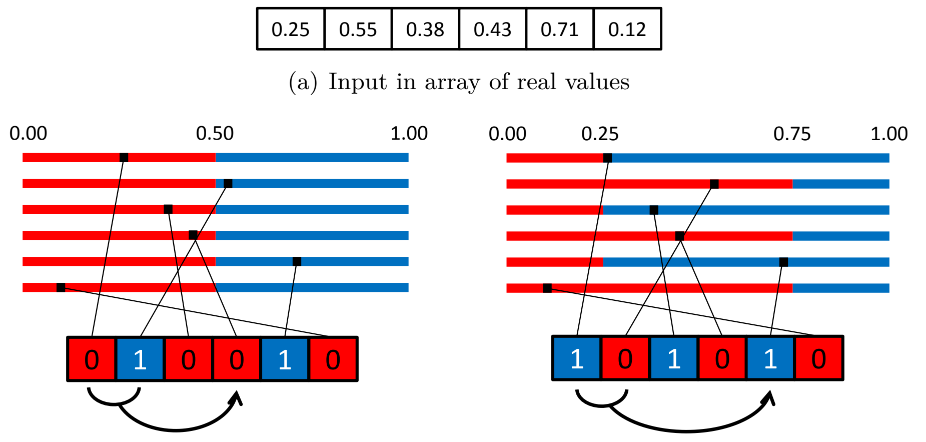
After presenting the results for binary multiplexer operations, the next comparisons involve assignments with real valued input where some preliminary results are available in [11]. The basic functionality of the input array is similar. The first k elements point out one of the rest, which contains the expected output. The obvious difference is that the inputs should be "translated" into binaries in order to determine the answer. This means, that the value in each element is either considered as a "0" or a "1". Other than this matter, the assignments are identical to the ones coping with binaries.

Environments producing real values have several flexibilities. In [37], Wilson introduces two tasks where each of them employs different sets of thresholds within a similar range [0.000, 1.000]. Table 5.3 shows that instead of using a static point at 0.500, employing a set of other threshold concepts (in this case, alternating) are also possible. This is a way to increase the complexity of a task, since covering a narrow range has a greater risk of creating coverage overlaps.

Table 5.3: Scenarios for real MP6 [37]

Task	Element	Range "0"	Range "1"
Constant threshold	All	[0.000, 0.500)	[0.500, 1.000]
Alternating thresholds	Odd	[0.000, 0.250)	[0.250, 1.000]
	Even	[0.000, 0.750)	[0.750, 1.000]

The difference between those scenarios can be illustrated by Figure 5.4, where an input containing the values ”0.25; 0.55; 0.38; 0.43; 0.71; 0.12” is fed to the system. When such state is translated using the static thresholds, it is interpreted as ”010010” and the correct answer is "0". Contrary to that, if an identical input is fed by the environment employing the alternating thresholds, the obtained binary version is ”101010” expecting an output "1".



(a) Input in array of real values

(b) Static threshold at 0.50

(c) Alternating threshold at 0.25 and 0.75

Figure 5.4: Binary interpretation types

Due to the unavailability of XCSR’s source code, the simulations only involves XCS-RC while the comparisons are made by using the experimental data from [37]. Both scenarios are run using similar parameter setting and alternating explore-exploit selection method to pick a winning action. The given results are presented only from the exploit trials, using a sliding window with a width of 100 data taken every 100 exploit trials. Both systems start all experiments without any prior knowledge.

The first comparison involving experiments with a constant threshold shows a slight difference on the performances. As depicted in Figure 5.5(a), the correctness rate of XCSR are superior for the first 4000 trials. Later between then and the 14 000*th* trial, XCS-RC performs a better learning rate. Finally, until the end of the simulation, both of them are relatively comparable with over 93% of the states are responded correctly.

Those results indicate that XCS-RC starts with a slower learning speed compared to XCSR. This is caused by the requirements of the RC technique to have a number of sufficient experienced classifiers before having the capability of producing general and accurate offspring. Good rules are important for making appropriate conclusions while also play significant roles in becoming disprovals for inadequate combining attempts. The lack of disproving rules potentially causes hasty conclusions and produces inaccurate classifiers, which will eventually be deleted after being executed. Such situation commonly occurs in the early phase of the learning process, especially when XCS-RC copes with an environment with a large state space. Once the system reaches an advantageous amount of knowledge, the performance gets significantly better. This is denoted by two indications which appear after the 3000*th* trial, i. e., the steep reduction of the population size and the improved correctness rate.

Regarding the population size, Figure 5.5(b) shows the capability of XCS-RC in minimizing the number of rules to below 100 after 37 200 cycles. For the whole simulations, the requirement of space is always kept less than 510 classifiers, never getting close to the limit at $N = 2000$. Further process shows that XCS-RC manages to reduce the size of $[P]$ below 25 rules after 220 000 trials. This is not provided in the graph due to the unavailability of XCSR's experimental data, which are only available for the first 40 000 cycles.

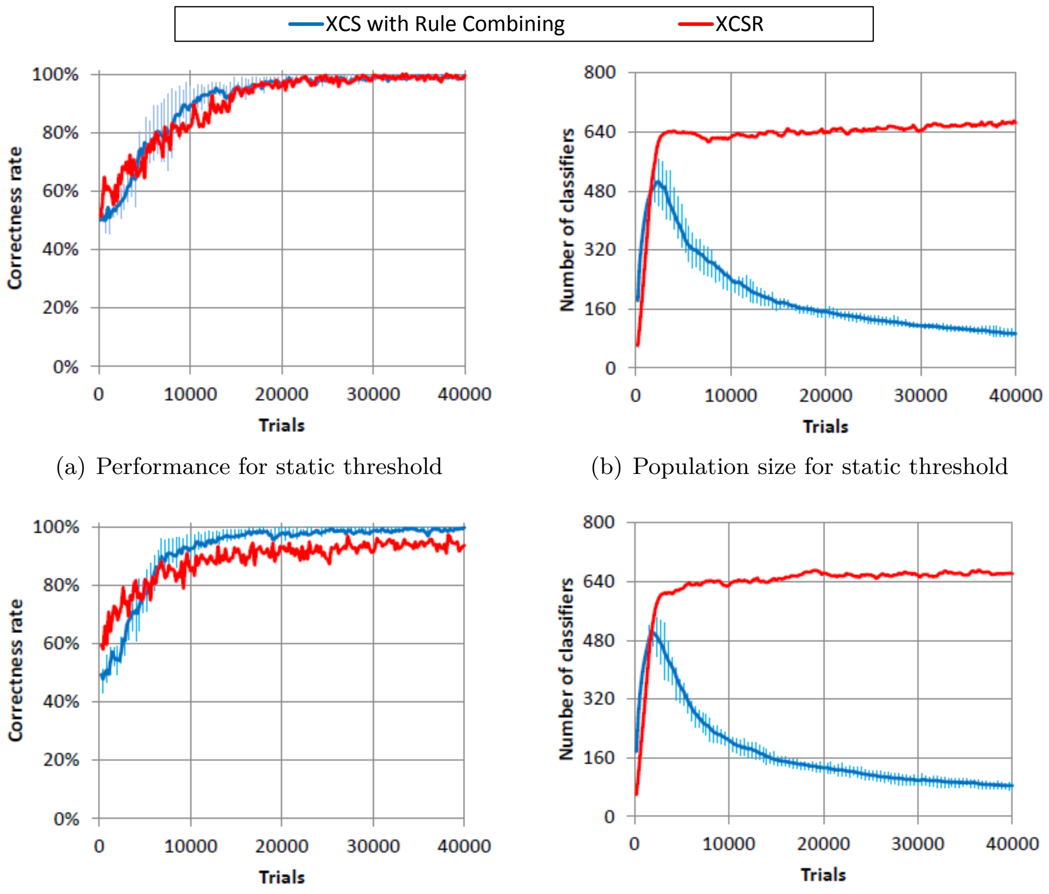
Figure 5.5(c) shows a comparison for the second experiment involving a case with alternating thresholds. XCS-RC does not show a significant change on the behavior, but XCSR gets some difficulties in coping with the new set of widths. Like in the previous task, XCSR is more superior for the first 4000 trials. However, starting from the 7000*th* cycle, RC assists the proposed system in performing a better learning capability. By the end of the simulations, XCS-RC presents a 99% of the full correctness rate while XCSR never reaches a 95% of a full performance for the whole simulations.

As in the experiment involving constant thresholds, XCS-RC is challenged by the lack of disproving rules in the beginning of the simulations. The population size is also identical to the previous test, i. e., always less than 510 individuals (see Figure 5.5(d)). The proposed system successfully manages to reduce the number of classifiers to below 90 after 40 000 trials, and further keeping the size less than 25 after 230 000 cycles. For XCSR, the achievement of its generalization attempt can be considered identical to that produced in the previous task.

Now, after observing these facts, one can say that XCS-RC is more powerful than both XCS in handling binary environments and XCSR for continuous inputs. To either strengthen or weaken such impression, some more difficult scenarios are provided. In the following subsection, the validation process continues by employing the checkerboard scenarios.

5.2.3 Checkerboard

The idea of using the so-called checkerboard problems to test classifier systems was introduced by Stone and Bull in [30]. It is a mapping task, where XCS learns to provide appropriate answers for all received states. The input consists of a number of elements containing real



(a) Performance for static threshold (b) Population size for static threshold

(c) Performance for alternating threshold (d) Population size for alternating threshold

Figure 5.5: Comparison for the real multiplexer tasks

values representing a specific coordinate within the environment. A checkerboard can be described as an m-dimensional cube where each dimension is divided into n areas. The name "checkerboard" came from a set of rectangles having alternating white and black colors similar to a checkerboard, when using a parameter setting $m = 2$ and $n > 2$.

A correct answer means that the learning system successfully identifies the color at the pointed coordinate. Figure 5.6 depicts an example using $m = 2$ and $n = 5$, mentioned as "CB-2d5" afterwards. An input consisting of two real numbers (i. e., 0.50 and 0.70) denotes a specific coordinate within a checkerboard. The former number belongs to the horizontal axis while the latter is interpreted vertically. Since (0.50, 0.70) in the checkerboard is located in a white area, the output is expected to represent that color (a simple term "WHITE" is used in this example).

Compared to the multiplexer, the challenge provided by this scenario is more difficult. Each dimension is commonly divided into more than two areas. Hence, the chance of overlapping is greater then the tasks of performing a multiplexer. As an example, Figure 5.7(a) depicts an XCS-RC's combining process involving two dots at the coordinates (0.42, 0.25) and (0.58, 0.31), represented respectively by cl_1 and cl_2. Since the classifiers own an identical action and similar predictions, the system attempts to combine them. The offspring enters $[P]$ after some usual examination procedures, covering a two-dimensional area shown in Fig-

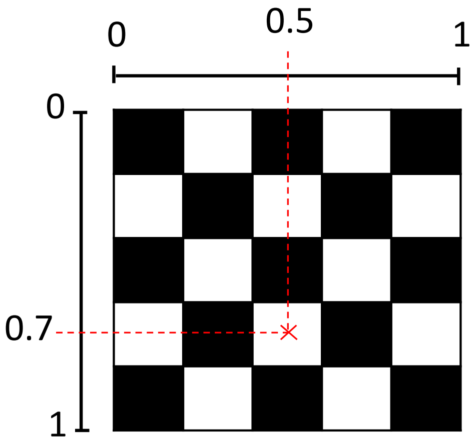


Figure 5.6: A mapping process in a $CB - 2d5$ scenario

ure 5.7(b)). This combining process is adequate, denoted by the non-overlapping coverage within the white cell.

A contradicting situation occurs when the parents currently cover two dots (or intervals) located in different cells. Figure 5.7(c) illustrates the combining process involving a pair of classifiers cl_3 and cl_4. As shown in Figure 5.7(d), the child's coverage overlaps into the black area. Although a series of examination is already conducted, the final result depends on any existence of disprovals. An inadequate combining can still be accepted when no disproving rule exists in the population. Commonly, such incident occurs at the early learning cycles and becomes more likely to occur when the value of T_{comb} is set too low.

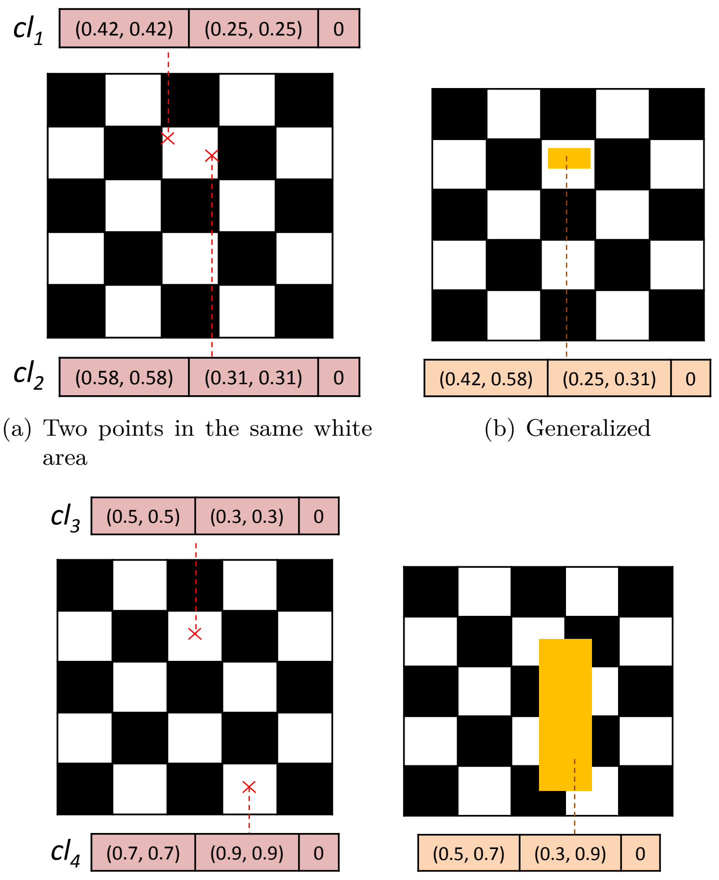
Setting the checkerboard with a greater number of dimensions certainly makes the problem more complex. A higher divisions decreases the area of each cell, and thereby increasing the chance of coverage overlaps. Both the number of dimensions and divisions affect the learning performance significantly. Assuming that all checkerboard scenarios involves environments having no less than three divisions, a checkerboard problem whose input consisting of six elements is theoretically more difficult to solve than a typical MP6.

Experiments involving XCS-RC are run using empty initial populations. The results are displayed using the average value out of 20 simulations. As in the previous cases, the correctness rate and the number of classifiers are recorded every 200 trials. The presented value for the performance is calculated by only considering the results from the last 100 exploit modes. The parameter setting is identical to that used in the real value multiplexer.

XCS-RC's performance in learning CB-2d3 scenario is shown in Figure 5.8(a). It successfully maps the environment into a set of rules. After 1200 trials, the line indicator shows no less than 90% of a full correctness rate. Furthermore, the system provides at least 99% of the full performance after the $5000th$ cycle.

In addition to the performance, the reduction attempt for the required space is successfully executed. XCS-RC aims at $[P]_{opt}$ for CB-2d3 which consists of 18 classifiers. In learning the given task, the system manages to produce 36 classifiers after 13 000 trials. Figure 5.8(b) depicts the requirement of space for keeping the knowledge, which is always kept below 20 after 47 000 cycles.

Similar results indicating the capability of XCS-RC in learning the checkerboards are shown in Figure 5.9. Obviously, CB-2d5 is more difficult to handle than CB-2d3. The complexity causes XCS-RC maintaining nearly 800 individuals at the beginning of the simulation before



(a) Two points in the same white area

(b) Generalized

(c) Two points in separate white areas

(d) over-general coverage

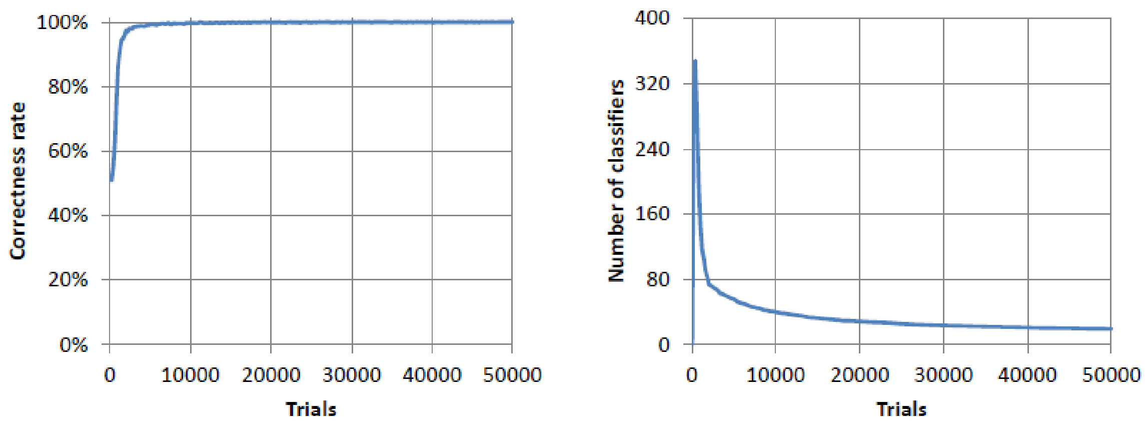
Figure 5.7: Combining in Checkerboard scenario

followed by a steep reduction afterwards (see Figure 5.9(b)). At the 20 000*th* trial, the population size is reduced to 81.5 rules before finally keeping only 52.5 classifiers at the end of the simulations. Since $[P]_{opt}$ for CB-2d5 consists of 50 classifiers, the provided result can be considered highly desirable.

Figure 5.9(d) illustrates the capability of XCS-RC in coping with a more difficult task, i. e., CB-2d7. The $[P]_{opt}$ for this problem consists of 98 individuals. Here, XCS-RC maps the environment using 164.85 rules at the end of the simulations. The population size is still unstable, which is an indication that the system requires more cycles to provide a better result. Nevertheless, the reduction can be considered significant, since the number of rules previously reaches beyond 900 classifiers at around the 2600*th* cycle.

However, a reduction of the population size is a secondary target after the performance. Regarding this matter, Figure 5.9(a) and 5.9(c) depict an impressive learning capability owned by XCS-RC. It never performs correctness rates at any value lower than 99% after 14 000 trials for CB-2d5, and after 32 000 cycles for CB-2d7. Although a longer run is stilll required to achieve a perfect result, these data confirm that XCS-RC has the ability to map various two-dimensional Checkerboard environments.

The last checkerboard test for XCS-RC is conducted by assigning a three-dimensional checkerboard, i. e., the CB-3d3 task. In [30], Stone and Bull attempt to solve a similar



(a) Performance (b) Population size

Figure 5.8: XCS-RC's results for the CB-2d3 problem

case using XCSR. Unfortunately, the results provided in the publication cannot be directly compared to RC technique, due to the non-identical initial states. The previous investigation uses a non-empty initial knowledge while XCS-RC always starts with zero classifier. Although a comparison would not be advantageous to XCS-RC, the results are mentioned here in order to get an impression of the proposed system's behavior.

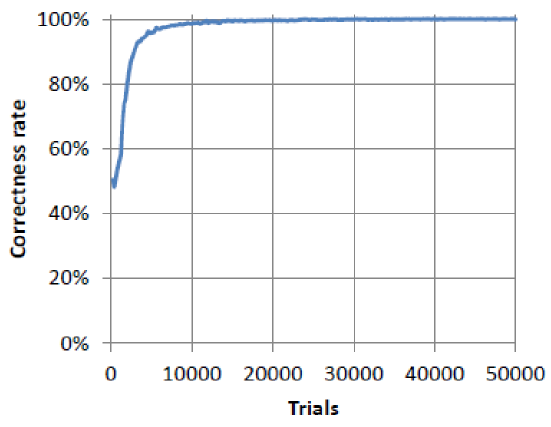
In solving CB-3d3, XCS-RC performs well by showing a capability of quicker learning compared to XCSR. Figure 5.10(a) shows that less than 11 000 cycles are required by the proposed system to get a stable correctness rate above 90%. It shows a superiority over XCSR that requires over 30 000 trials to get a level performance. Then, after 35 000 trials, XCS-RC never performs below 99% correctness rate. At the other hand, XCSR is unable to reach such achievement even after 200 000 learning cycles, which marks the end of the simulations.

The population size limit for XCS-RC is set at $N = 2000$, equal to the amount of XCSR's initial rules. Figure 5.10(b) shows that by employing RC technique, the number of $[P]$ members never exceeds 600 classifiers. Then, the proposed system keeps the number of rules below 200 after 24 200 cycles and ends with 101 individuals. Such combination of successful achievements in performance and rule reduction is contrary to that belongs to XCSR, which copes with 600 classifiers after 200 000 trials (see [30]).

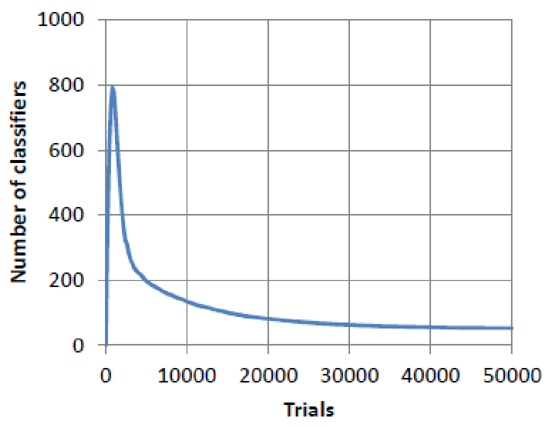
As a summary for the Checkerboard tests, XCS-RC can be considered capable of mapping the environment successfully. Although a comparison is merely possible, one can argue that RC assists the learning system to tackle the assigned tasks. Using an empty initial knowledge, the given performance is quite convincing while the population size is reduced significantly. Specifically in facing the CB-3d3 case, XCS-RC can be considered more superior to XCSR, despite the difference regarding the initial knowledge. Now, the validation process continues to test the proposed learning system using some binary multi-step problems. Then, the result is compared to those produced the original XCS to observe the effect of the modification, as presented in the following subsection.

5.2.4 Woods and Maze

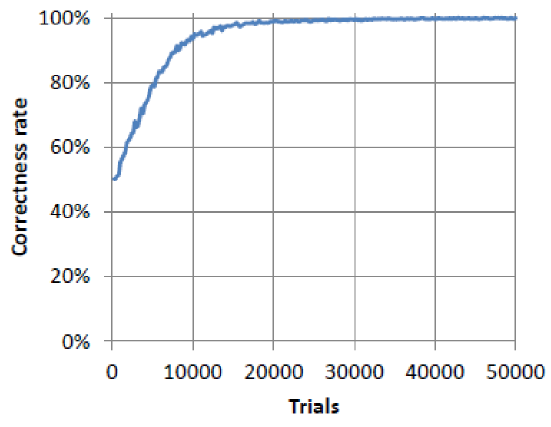
The tests validating the capability of XCS-RC in solving single-step problems are followed by a series of investigations involving binary multi-step tasks. The difference between those



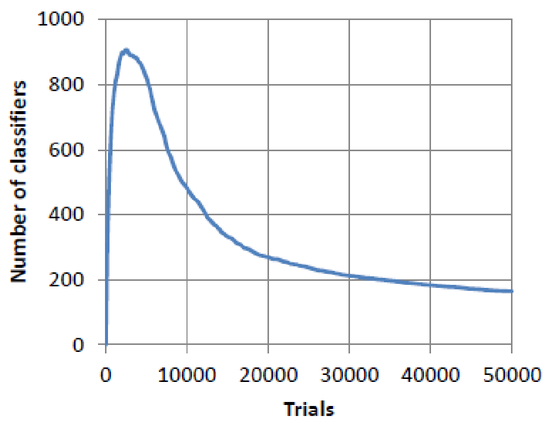
(a) Performance (2d5)



(b) Population size (2d5)



(c) Performance (2d7)



(d) Population size (2d7)

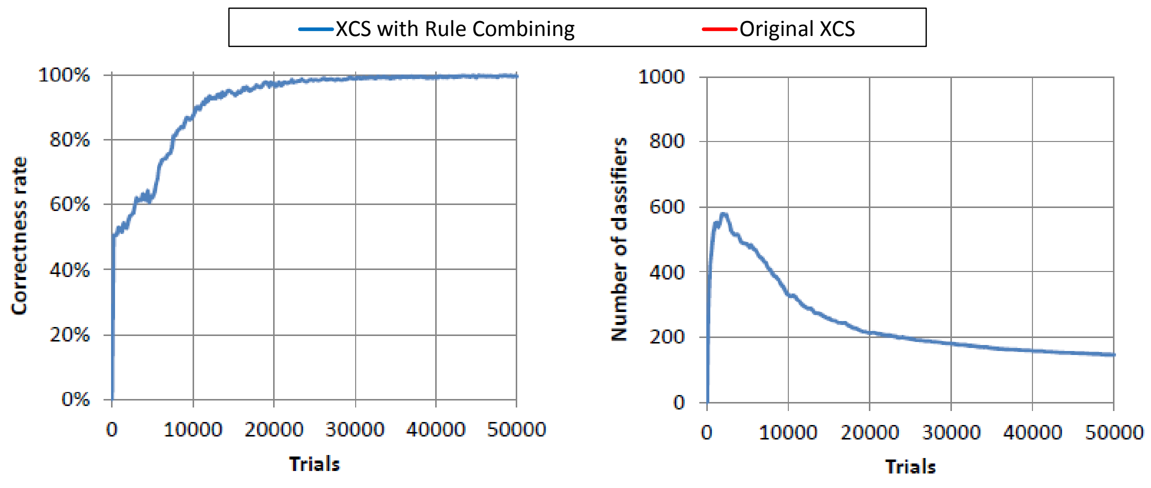
Figure 5.9: XCS-RC's results for the CB-2d5 and CB-2d7 problems

environments is the feedback mechanism. In a single-step problem, each system's action gets an instant reward. Different to that, in a multi-step case, executed rules are collected and the environment sends a feedback when a specific situation occurs. The investigations presented here employ various artificial Markovian environments namely *Woods* and *Maze*.

Firstly, a brief explanation regarding Markovian environments is provided here. Figure 5.11(a) and 5.11(b) depict two environments namely Woods1 and Woods101, respectively. The former is a Markovian while the latter is not. A non-Markovian environment is denoted by an existence of two or more empty cells having identical Moore neighborhood (the eight surrounding cells), which in this case is the ones marked with a cross.

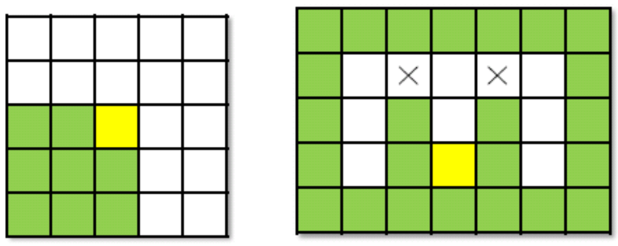
Each map contains a number of cells with various colors (i. e., white, green and yellow). The white color denotes emptiness, a green cell means rock and a yellow one indicates the availability of some food. Some other environments are designed to have a higher complexity (e. g., Woods2) by adding two additional colors Orange (alternate food) and Dark Green (alternate rock). Woods1 also has a characteristic of being a torus world, meaning that a cell at any of its edges is connected virtually to the opposing edge. Regarding this matter, one can easily conclude that Woods101 is not a torus environment.

A simulation begins by placing a virtual entity called *animat* at a random empty cell. At



(a) Performance (b) Population size

Figure 5.10: Results for the CB-3d3 problem



(a) Woods1 (Markovian) (b) Woods101 (non-Markovian)

Figure 5.11: Maps of the Woods1 and Woods101 environments

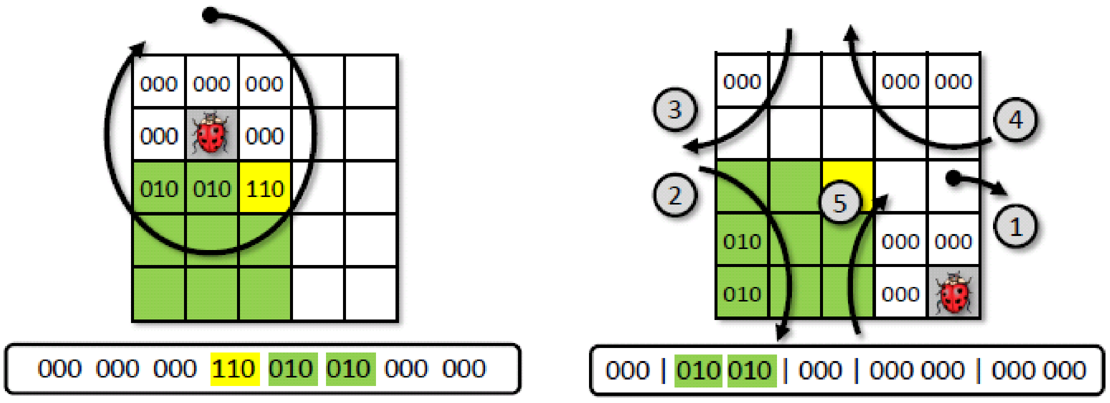
every time step, the animat picks an action to move to a cell within its Moore neighborhood either horizontally, vertically, or diagonally. An option to not moving is unavailable, but a decision to move towards a rock causes no change of location. The rules representing a series of animat's movements are collected, and the trial ends when the animat arrives to the food. Then, a reward is given and another trial is conducted by placing the animat at a random empty cell.

The animat's task is to get closer to the food and "eat" it while attempting to minimize the number of steps. In order to do that, the animat observes its Moore neighborhood and encodes the observation result into binaries. The obtained code is fed to the classifier system, which is used by the animat to learn the assigned task. Table 5.4 provides a complete list of cell types and the corresponding codes.

Figure 5.12(a) illustrates an encoding process, where an animat translates its Moore neighborhood into a 24-bit array of binaries. Starting from the northern cell, the animat observes the cells clockwise and encodes each of them while composing the input. Since Woods1 is a torus world, it is still possible to form a 24-bit input while located at the edge of the environment, as depicted in Figure 5.12(b). Here, each vertical line between the binary codes simply visualizes a different arrow representation, without any actual effect to the "normal" input.

Table 5.4: Binary codes for the cells in Woods and Maze environments

Type	Color	Code
Empty	White	000
Food	Yellow	110
	Orange	111
Rock	Green	010
	Dark Green	011



(a) At a cell in middle (b) At the edge

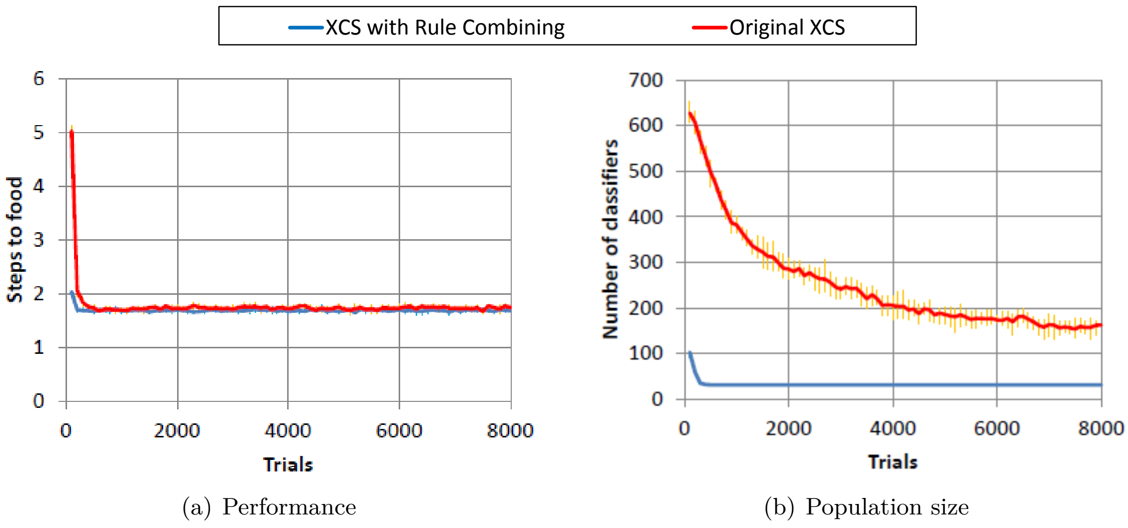
Figure 5.12: Input composition process by the animat

Such input composition is followed by the usual mechanism of the classifier system, from matching to executing the chosen action. The difference comes afterwards, i. e., instead of sending an instant feedback, the environment waits until the animat gets the food. When that happens, a reward is given and distributed to the collection of executed classifiers. In conclusion, the more steps required for getting the food, the smaller reward received by each individual. Hence, the performance of the system in coping with such scenario is determined by its ability to minimize the number of steps.

Since the composed input does not involve any information regarding the animat's past movements, its decision is solely based on the information of the current surrounding cells. Such design makes a non-Markovian environment unsuitable for performing a multi-step learning. Using Woods101 as an example, when the animat is located in one of the cells marked with a cross, both options "WEST" and "EAST" have an equal chance to be either the best solution (closer to the food) or the worst option (further away).

In the provided graphical presentation, the system performance is determined by the average number of steps out of 20 runs, which should be minimized. The expected value can be calculated by taking a distance-weighted average from all empty cells. E. g., among 16 empty cells in Woods1, 11 of them are two steps away from the food while the other five requires only one. So, the expected number of steps is $(2 \times 11 + 1 \times 5) \div (11 + 5) = 1.6875$. In conclusion, a learning system is considered successful by providing an average value at around 1.6875 steps.

The following multi-step simulations follow the parameter setting used in [36]. For XCS-RC, the default values are used except $predErrTol = 5$. All runs still use the typical alternating explore/exploit regime, where the graphical representation is provided by only considering the results from the exploit trials.



(a) Performance

(b) Population size

Figure 5.13: Results for the Woods1 environment

Both XCS and XCS-RC are able to map the Woods1 with an average close to 1.6875 steps after performing 200 trials (see Figure 5.13(a)). However, Figure 5.13(b) indicates a significant difference between their generalization capabilities. XCS-RC successfully reduces the population size down to less than 36 classifiers after 300 trials. In contrast, XCS maintains a population of approximately 160 rules at the end of the simulations.

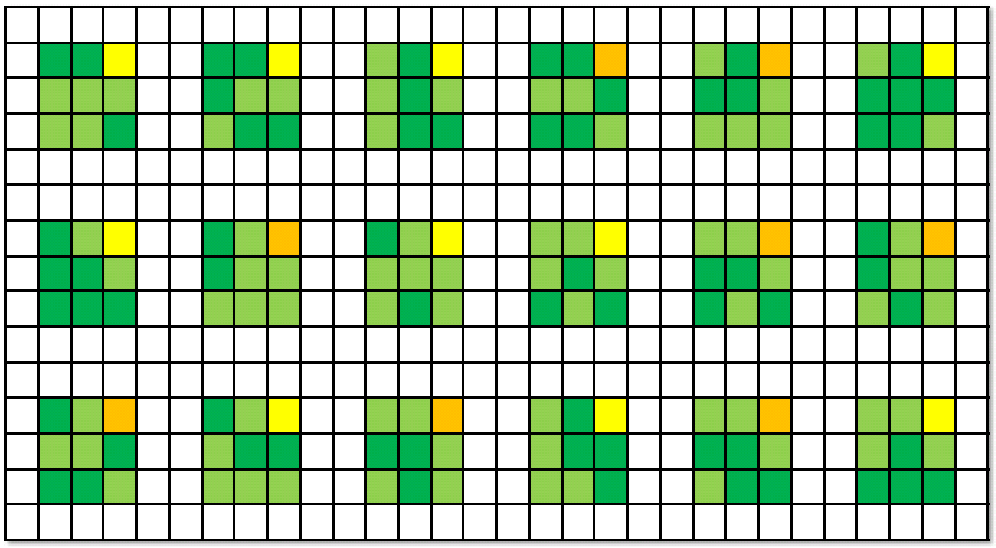
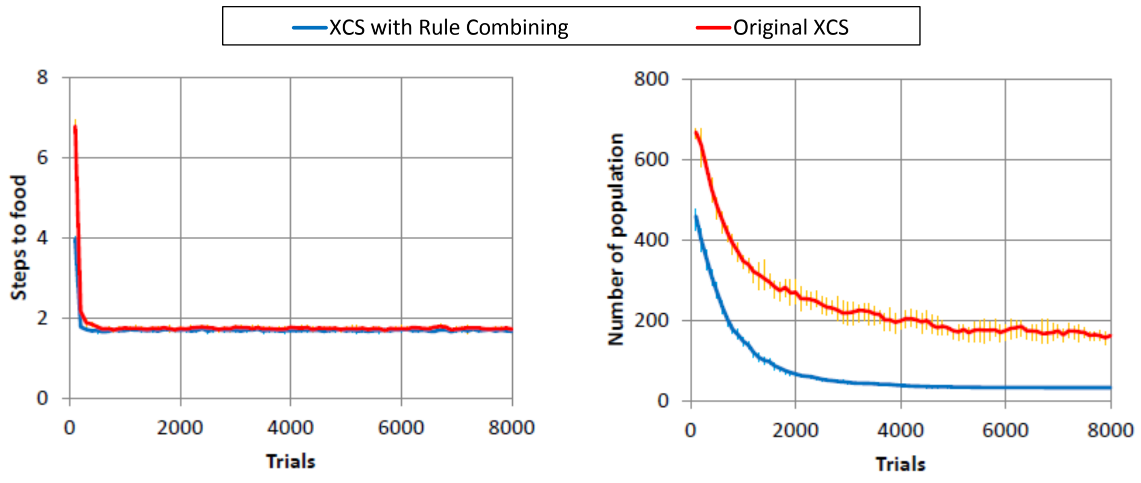


Figure 5.14: Woods2 environment

The next test involves a Markovian environment namely Woods2, as depicted in Figure 5.14, which is also a torus one. Compared to Woods1, the complexity of this case is increased by two additional colors, i. e., Orange and Dark Green. The former represents some food while the latter symbolizes a rock. Despite having different translation codes (see

Table 5.4), the characteristic of an orange cell is identical to those in yellow, while dark green is similar to green.

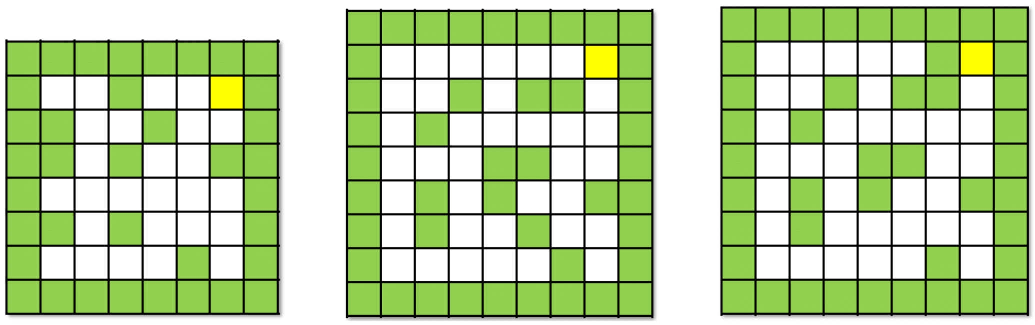


(a) Performance (b) Population size

Figure 5.15: Results for the Woods2 environment

After placing the animat at an empty cell at the beginning of each simulation, the number of steps to food is monitored. Then, the average is recorded every 50 exploit trials with an performance at around 1.6875 steps. Figure 5.15(a) shows that both XCS and XCS-RC are able to manage the number of steps close to that value, in less than 400 trials. However, their generalization capabilities show a significant difference.

XCS-RC maintains a population size lower than 40 after 4000 trials. Contrary to that, XCS still struggles with around 160 rules at the end of the simulations. Despite the chance that XCS might reach such achievement in the future, XCS-RC requires a significantly less cycles to accomplish the assignment. In conclusion, the RC technique provides a notable assistance to the classifier systems in facing some cases involving the Woods environments.



(a) Maze4 (b) Maze5 (c) Maze6

Figure 5.16: Maps of the Maze4, Maze5 and Maze6 environments

Further on the multi-step test, the challenge is increased by assigning the learning systems to map Maze4, Maze5 and Maze6 environments (see Figure 5.16). Maze5 and Maze6 are employed as test cases in [20], where Lanzi introduces an XCS variant with an additional component called *Specify.* The proposed functionality enables XCS to specialize rules, as opposed to generalizing them. Such technique is implemented to the classifiers that are indicated as over-general. Unfortunately, the results from that investigation cannot be compared directly to the ones performed by XCS-RC since Lanzi equips his XCS with an initial population. However, mentioning the provided results is still useful to get a rough impression.

In coping with Maze environments, some parameter adjustments are applied to XCS, i. e., $P_{\#} = 0.05$ and $N = 2000$. Experiments show that a greater wildcard probability may cause failures, as well as a lower population size limit. Then, using a set of identical procedures to the Woods in running the simulations, Figure 5.17(a) is provided as the result of the experiments. There, both XCS and XCS-RC are indicated to have comparable performance in mapping Maze4, close to the expected value at 3.50.

Regarding the population size, XCS indicates a failure in reducing the number of rules. The original system still owns an average of 601 classifiers at the end of the simulations. Contrary to that, XCS-RC quickly reduces its population to a size of less than 120 rules in only about 900 trials. As shown in Figure 5.17(b), XCS-RC confirms its superiority to XCS in terms of generalization.

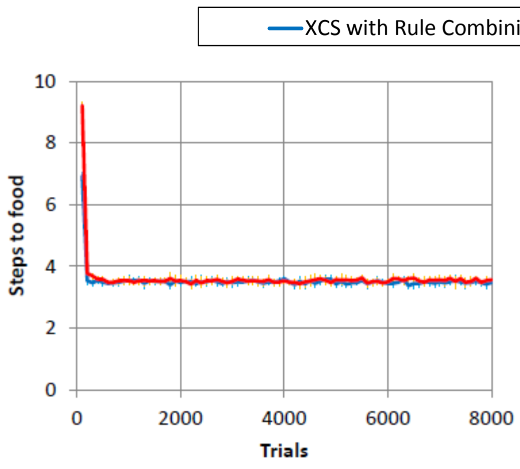
The results from the experiments involving Maze5 and Maze6 are similar to those belong to Maze4. Both learning systems show successful attempts in handling the assignments. XCS-RC requires 200 trials to minimize the number of steps, reaching around the expected value at 4.6111 for Maze5 and 5.1944 for Maze6. In comparison to XCS, the original system performs a comparable behavior with a slightly higher number of steps for Maze5 and a similar achievement for Maze6.

However, Figure 5.17(d) and 5.17(f) indicate the incapability of XCS to reduce the population size at a similar level compared to XCS-RC. The proposed method manages to decrease the number of rules to less than 200 after 1300 trials for both environments. At the other hand, XCS ends the simulation with around 840 and 720 classifiers for Maze5 and Maze6, respectively. Despite the chance of having a different situations in longer runs, XCS-RC shows a quicker generalization compared to the original XCS.

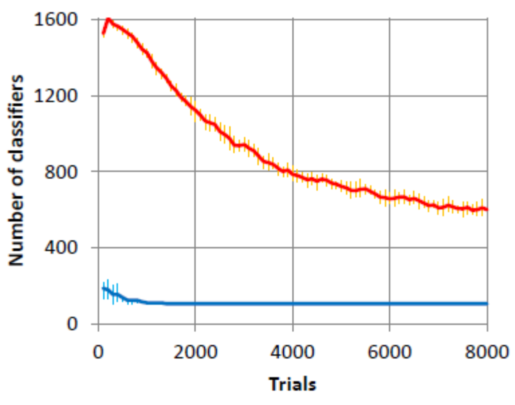
In the investigations involving Maze environments, Lanzi proposes a variant called XCS with Specify (XCSS) [20]. The modified system performs a more powerful learning capability, indicated by the results in coping with the given cases. After approximately 400 exploit problems (800 trials), XCSS requires less than 10 steps to reach the food in Maze5, and less than six steps in Maze6. In comparison, XCS-RC manages to perform such achievements after 300 trials (150 exploit problems). Another difference between XCSS and XCS-RC is that the former starts the simulations with an initial population while the latter has an empty one.

As a conclusion, one can argue that XCS-RC is more superior to XCS in performing generalizations to Markovian environments, represented by those Woods and Maze. Despite the fact that both provide a comparable learning rate, the original system requires a significantly more trials to produce general and accurate rules. In conclusion, RC technique performs better compared to the Darwinian genetic operators.

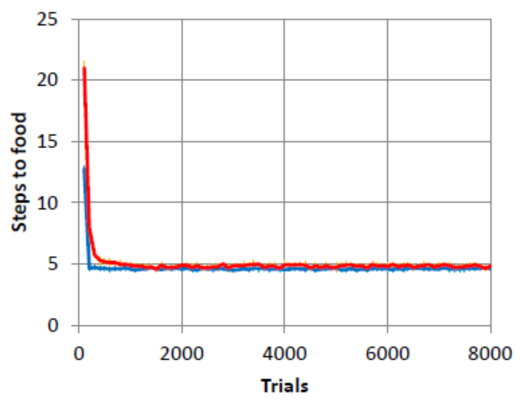
The investigations using Markovian environments as the test cases end the series of comparisons between XCS and XCS-RC. Although the proposed system performs a desirable behavior in most of the tests, some interesting drawbacks are shown, as discussed in the following section.



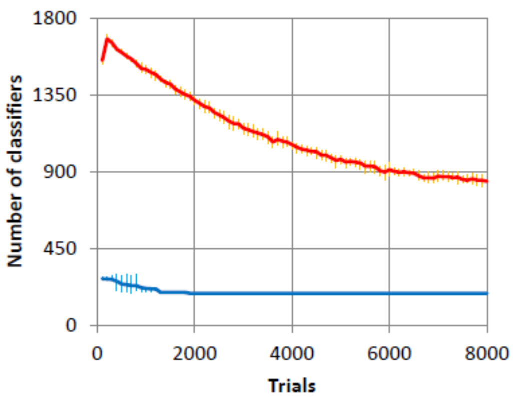
(a) Performance for Maze4



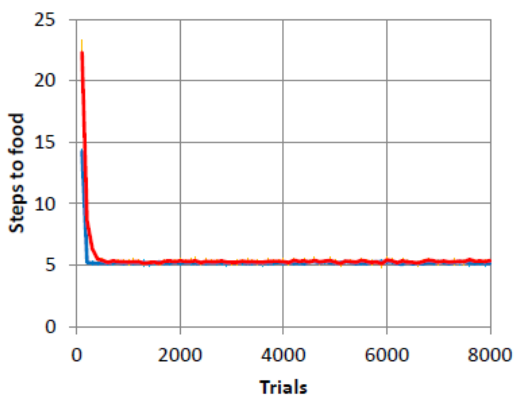
(b) Population size for Maze4



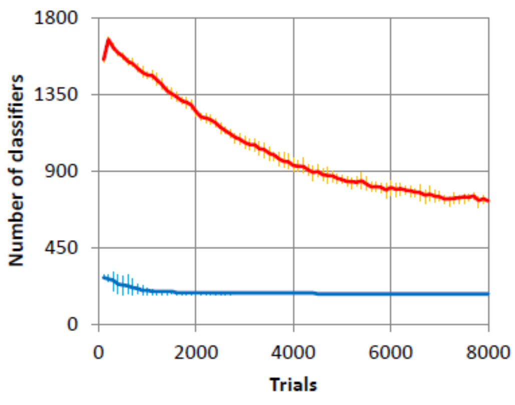
(c) Performance for Maze5



(d) Population size for Maze5



(e) Performance for Maze6



(f) Population size for Maze6

Figure 5.17: Results for the Maze4, Maze5 and Maze6 environments

5.3 Evaluation

Through a series of comparisons between XCS-RC and XCS, a clear conclusion can be made that the proposed system is more superior than the original XCS and some of the published variants. The implementation of the induction concept is proven to have a powerful assistance in all experiments involving multiplexers, checkerboards and Markovian environments. XCS-RC convincingly provides a better learning rate and a quicker discovery of general and accurate rules.

The conducted experiments also show some indications of drawbacks, affecting the performance of XCS-RC. The first notable weakness is loss of knowledge. This occurs when XCS-RC detects a sign of hastiness after updating the rules in $[A]$. When the flag indicating over-generality rises, the corresponding rule is deleted. Such step is necessary since keeping hasty offspring potentially drives the system to making incorrect decisions. However, each removal definitely causes some knowledge permanently being eliminated.

The second evaluation is regarding the execution time. Experiments show that XCS-RC requires less than one second to solve a simple task, e. g., the binary MP6 simulation with 10 000 trials. Obviously, more cycles are required for learning some more complex cases, e. g., binary MP20 with 200 000 trials. Running a series of binary MP20 using only 10 000-trial simulations does not satisfy the purpose of the experiment.

Finally, the last topic presented in this section is focused on the parent-picking mechanism. Although XCS-RC shows its capability to learn all the given cases, only a few of them successfully reach $[P]_{opt}$. A series of observations shows that such failure is caused by the inability of the system in determining which pair should be prioritized to be combined. These evaluations are discussed in more details, as follows.

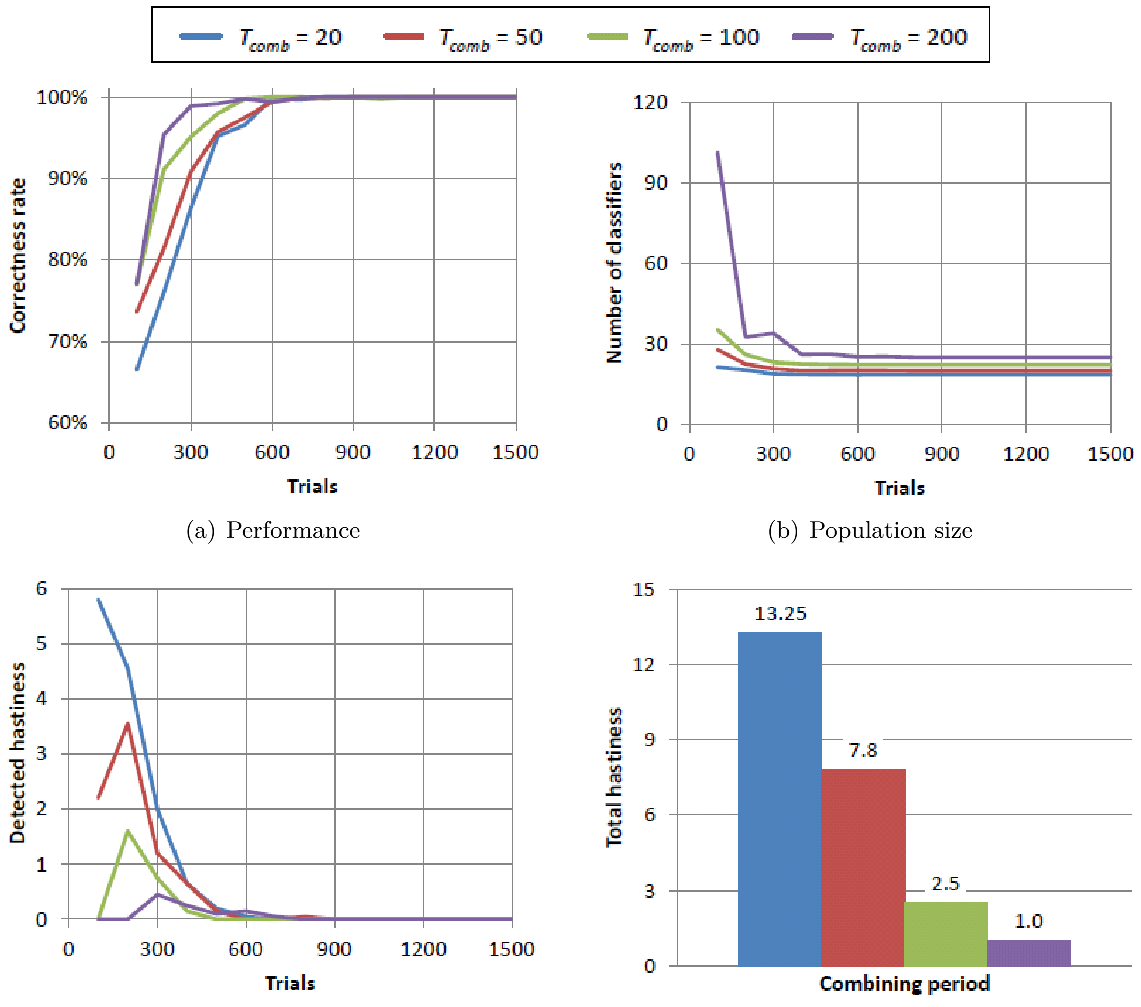
5.3.1 Loss of Knowledge

In the attempt of performing a generalization, XCS-RC takes some values owned by a number of existing rules and creates an offspring. Ideally, all involved rules are accurate and coming from a population that contains a comprehensive knowledge, including potential disprovals. However, the possibility that some children are created inadequately due to the lack of disproving classifiers remains open. To anticipate the negative impact of such disadvantageous combining, XCS-RC is equipped with a deletion mechanism. The procedure removes rules being identified as a product of a hasty generalization.

However, such deletion does not only remove a classifier. It also eliminates the parents and all subsumed rules, which might be accurate ones. Those classifiers are deleted when the child is inserted to the population. Hence, deleting over-general offspring means taking a risk of also removing some lessons kept in a number of accurate individuals.

To observe how such situation may affect the results, a series of experiments is conducted to XCS-RC using a set of combining period T_{comb} with varying values. The rest of the parameters are set identical to those in the defaults, and the provided graphics are averaged out of 20 runs. Figure 5.18(a) depicts the result of the simulations involving binary MP6 with varying combining period values.

Since binary MP6 is a very simple problem, the difference of the results cannot be distinguished easily, due to the insignificance. However, a trend is shown indicating that the blue line representing $T_{comb} = 20$ has the slowest learning rate. The purple line with $T_{comb} = 200$ can be considered as the quickest. The red line with $T_{comb} = 50$ lays between those lines having combining periods 20 and 100 cycles.



(a) Performance (b) Population size

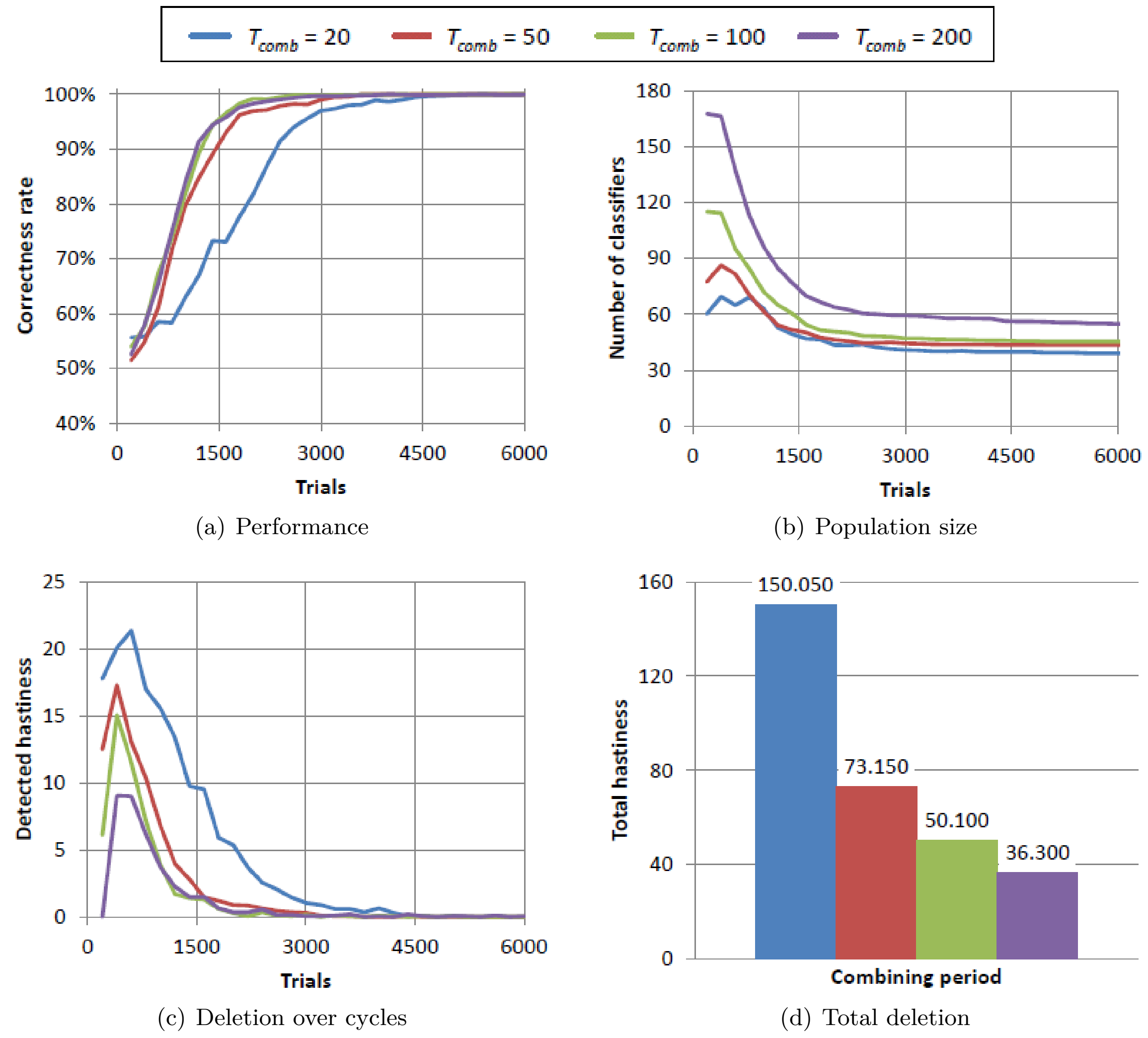
(c) Deletion over cycles (d) Total deletion

Figure 5.18: Comparison of results involving MP6 with varying T_{comb} values

The evaluation of the proposed system is also based on the population size, as shown by Figure 5.18(b). Although setting $T_{comb} = 200$ can be considered as the most desirable in terms of learning rate, an observation to the generalization capability states the opposite. The purple line shows that the system has not perform any generalization at the first data acquisition. In other words, the provided correctness rate comes from a set of ungeneralized classifiers. Therefore, in terms of population size, an exactly reversed sequence is presented. The purple line shows the least desirable generalization capability, while the blue one ($T_{comb} = 20$) can be considered as the winner.

Figure 5.18(d) explains the cause of the performance shown by the system. A smaller value of T_{comb} tends to having a higher number of deletions due to hastinesses. Then, Figure 5.18(c) confirms the number of removals of over-general rules by presenting the detected hastiness over cycles. The green and purple lines with a respective T_{comb} value at 100 and 200 cycles, show a similar zero deletion until the 100*th* cycle. This is due to the fact that the corresponding runs have not perform any combining yet. The blue has the highest deletion rate, while the red is the second best.

Those results can be interpreted this way: a system with a lower combining period tends to combine more rules, taking risk of performing hasty generalizations. Hence, it performs a relatively lower learning rate but maintains a smaller population size. To confirm the conclusion, another series of simulations using MP11 are run, with a similar set of combining periods.



(a) Performance

(b) Population size

(c) Deletion over cycles

(d) Total deletion

Figure 5.19: Comparison of results involving MP11 with varying T_{comb} values

A significant difference of performance is shown by XCS-RC with $T_{comb} = 20$, depicted by the blue line in Figure 5.19(a). It indicates a lower learning rate compared to the other systems which perform quite similarly. Such phenomena is caused by the level of complexity owned by an MP11 scenario. Handling the task using a low combining period is inadvisable. Therefore, solving the MP11 scenario with higher values of T_{comb} as depicted by the other lines, is more adequate.

Figure 5.19(b) shows a graphical representation of the population size for the simulations. Similar to that belongs to MP6, the number of classifiers depicted by the purple line is the greatest of all, i. e., around 56.00 individuals after 5000 trials. The green and the red, with a respective combining period set at 100 and 50 cycles, depict similar results. Then, the blue line provides the smallest number of rules along the simulations, although it also shows the lowest learning rate.

The cause of such slow performance is shown by Figure 5.19(d). With an average of over 150 rules deleted due to the occurrence of hastiness, the system with $T_{comb} = 20$ loses a relatively great amount of knowledge. It requires around 4400 cycles to reach a considerably low removal rate (see Figure 5.19(c)) and approximately the same time to perform a correctness rate consistently above 99%.

In conclusion, the results suggest that there is a strong relation between the number of hasty generalizations and the system performance. It can also be concluded that using $T_{comb} = 20$ is inadvisable for solving an MP11 problem or other tasks with a higher complexity. Systems with other values of combining period perform similar trends the previous task involving MP6. A higher T_{comb} value leads to a lower number of hastiness.

These results also emphasize the necessity of adjusting the combining period following the characteristics of the assignment. A complex problem requires more individuals before performing its first combining attempt. By assigning a greater value, the loss of knowledge can be minimized. At the other hand, there is a payoff between the system performance (which is affected by the number of hastiness) and the population size. The number of rules in the population tends to be greater when the system employs a longer period for performing RC. Fortunately, evaluating a periodic parameter such as T_{comb} is easier than dealing with a set of probability parameters to perform randomizations.

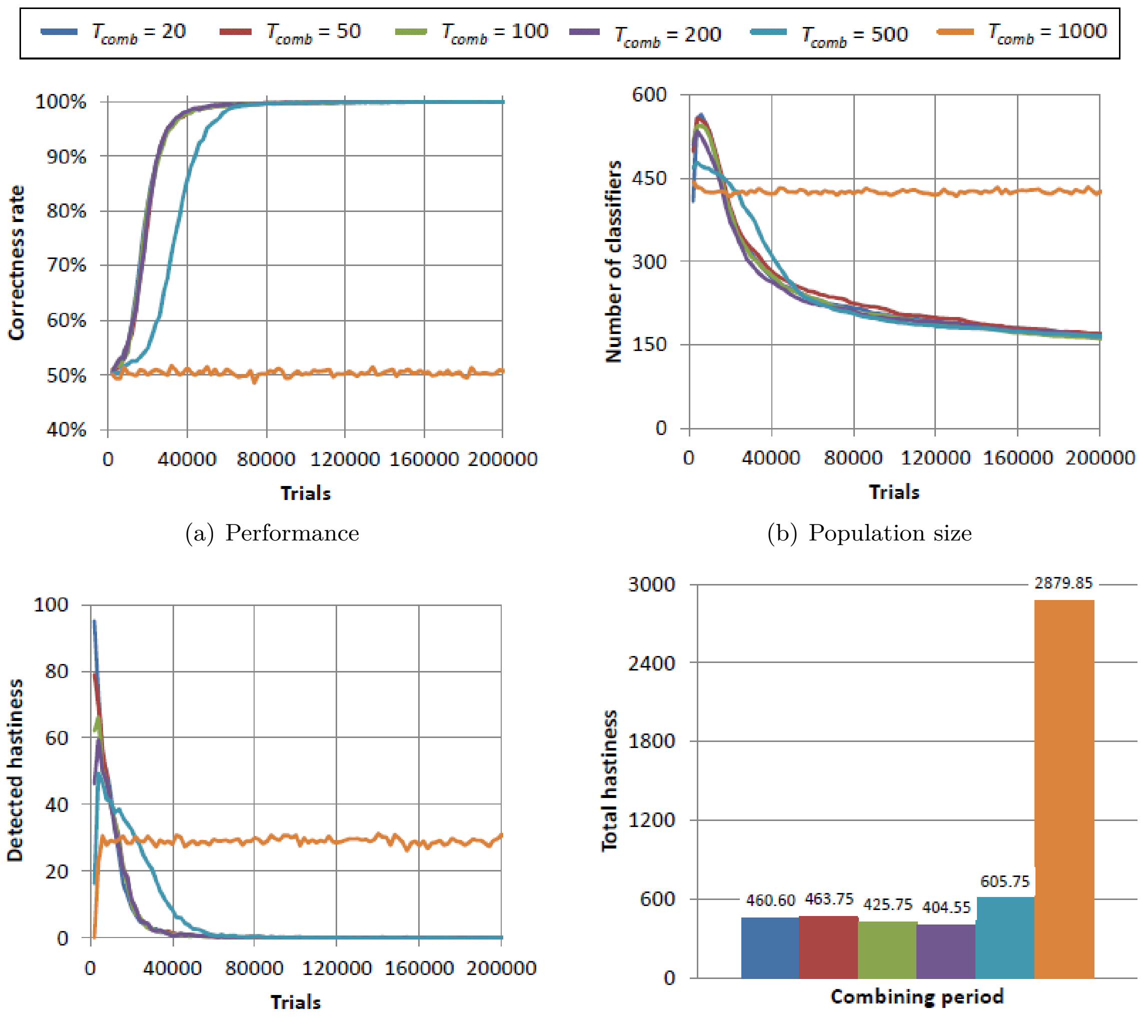
5.3.2 Runtime

Discussing the runtime is particularly required to evaluate XCS-RC in solving cases with a large input space, e. g., binary MP20. The obtained results indicate that the determining variable for this issue is the maximal population size parameter N. A lower threshold tends to make the proposed system perform quicker than a higher one. To observe the effect in more details, a series of simulations are run and the results are presented using the following comparison.

Figure 5.20(a) displays the performance of XCS-RC with $N = 800$ and varying T_{comb} values. Most lines indicate a similar behavior, i. e., a stable correctness rate above 95% after around 32 000 trials. Then, the light-blue line with $T_{comb} = 500$ performs a lower learning rate by requiring 50 000 trials to reach such level of performance. Finally, a learning failure is shown by the brown line ($T_{comb} = 1000$) with around 50% of full performance along the simulations.

Figure 5.20(b) and 5.20(c) depict similar results indicating better system capabilities of tackling the given case. Also, the light-blue line shows a slightly less desirable performance while the brown line fails to provide a good one. The population size owned by the system with $T_{comb} = 1000$ stays at around 425 classifiers for the whole simulations. Moreover, its deletion rate is very high with a total of slightly under 2879.85 hastinesses (see Figure 5.20(d)).

Such failure in coping with the task are caused by the space insufficiency. Assuming all input states are unique, the system already keeps 800 classifiers at the $400th$ trial. Some rules must be removed afterwards to keep the amount below the allowed limit. This means, for every simulation using $T_{comb} > 400$, a series of deletions is already executed before the first RC operation. Assuming the worst case scenario that all inputs are unique, XCS-RC with $T_{comb} = 1000$ would have to remove 1200 classifiers before the first generalization attempt. From those amount, at least 200 of them are experienced and accurate with a potential of disproving inadequate generalizations.



(a) Performance (b) Population size

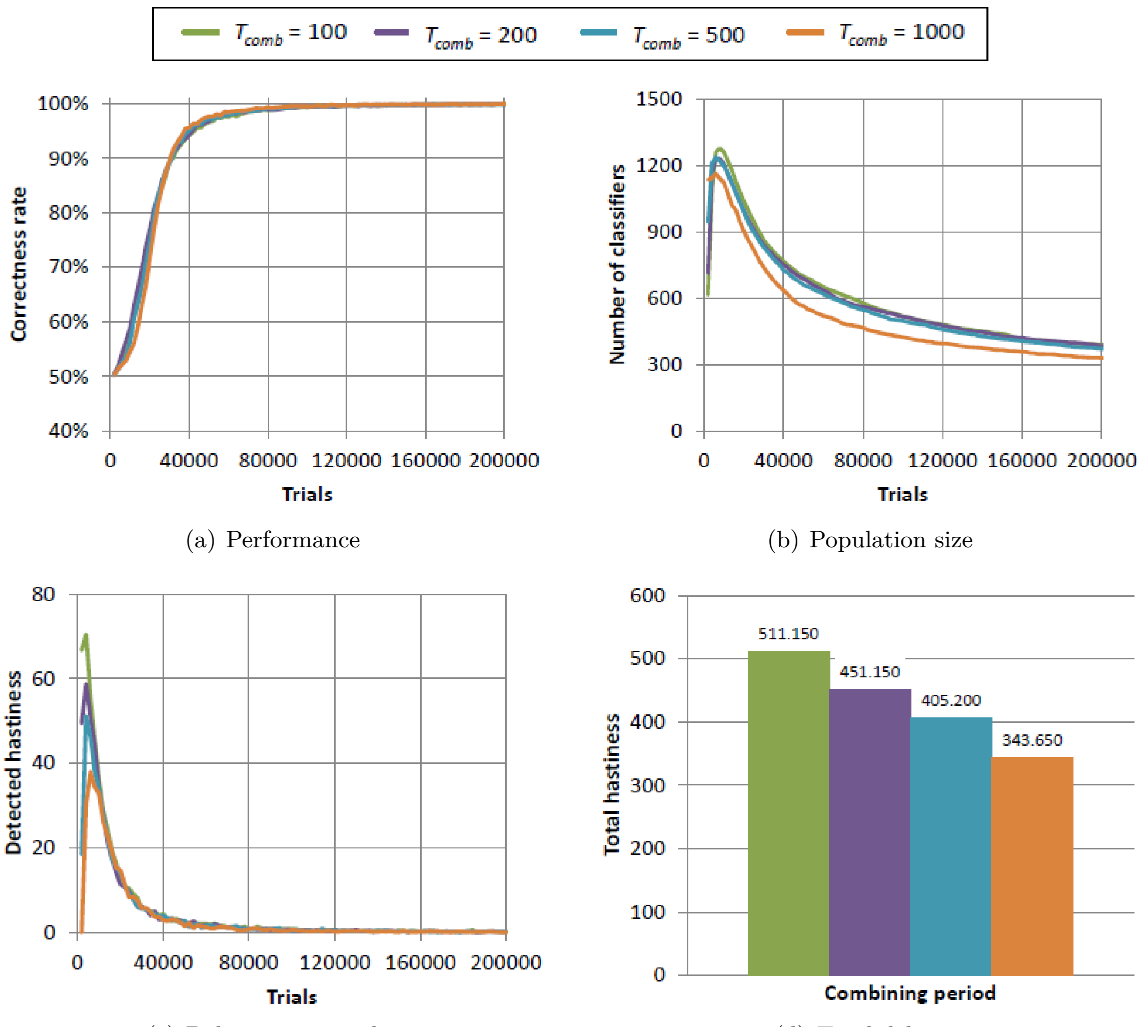
(c) Deletion over cycles (d) Total deletion

Figure 5.20: Comparison on MP20 using $N = 800$ and varying T_{comb} values

The space issue can be solved by increasing the size limit for the population. Therefore, another series of simulations using $N = 2000$ are run, particularly for those having $Tcomb = 100$ or greater. Figure 5.21(a) depicts the results where all systems perform at approximately the same level. Then, Figure 5.21(b) shows that the brown line representing XCS-RC with $T_{comb} = 1000$ perform the most desirable generalization capability. After the $160\,000th$ cycle, it maintains only around 360 classifiers while the others still cope with at least 400 rules.

Regarding hastiness detection, XCS-RC with $T_{comb} = 1000$ removes a total of 343.65 classifiers, averaged from 20 runs (see Figure 5.21(d)). This represents the lowest number of hasty generalizations compared to the others, despite that most of the difference occur at earlier cycles, as depicted by Figure 5.21(c).

In conclusion, using a greater N value increases the system capability of handling tasks with a large space input. XCS-RC is unable to solve a binary MP20 case using $T_{comb} = 1000$ and $N = 800$. At the other hand, changing the population size limit to 2000 provides slightly more desirable results. However, the ability of handling problems having a large space does



(a) Performance (b) Population size (c) Deletion over cycles (d) Total deletion

Figure 5.21: Comparison on MP20 using $N = 2000$ and varying T_{comb} values

not necessarily mean that the system is more superior, particularly when the runtime factor gets involved.

A short description regarding the execution of RC mechanism is presented before comparing the results. As seen in Figure 5.22, a generalization attempt is executed every T_{comb} trials. Therefore, every time the number of cycles reaches T_{comb}, RC is performed and the count is reset back to zero.

In practice, the system examines a flag that indicates whether the content of $[P]$ is changed since the last RC attempt. The aim is to prevent unnecessary time consumption, which will occur such as when the system already reaches a stable population (e. g., $[P]_{opt}$). This mechanism makes the system more efficient and requiring a shorter runtime. If the flag is up, a normal RC process is executed. Afterwards, XCS-RC runs a series of generalization procedures as explained in Chapter 3.

Two processes are taken into account when determining the runtime variable. The first one measures the decision making process, from receiving an input to executing the chosen action. Then, the second time measurement begins when a feedback is received from the environment

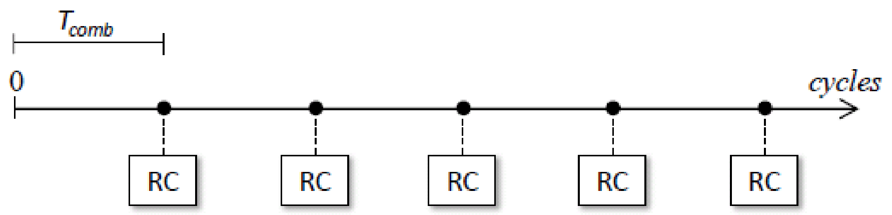


Figure 5.22: Execution of RC at every T_{comb} cycles

and covering rule update, occasional combining, and finally ends after the population is updated. All simulations are run in the same machine, one at a time. An identical set of 20 random seeds in provided to the system, and the measurement is determined by calculating an average of those 20 runs.

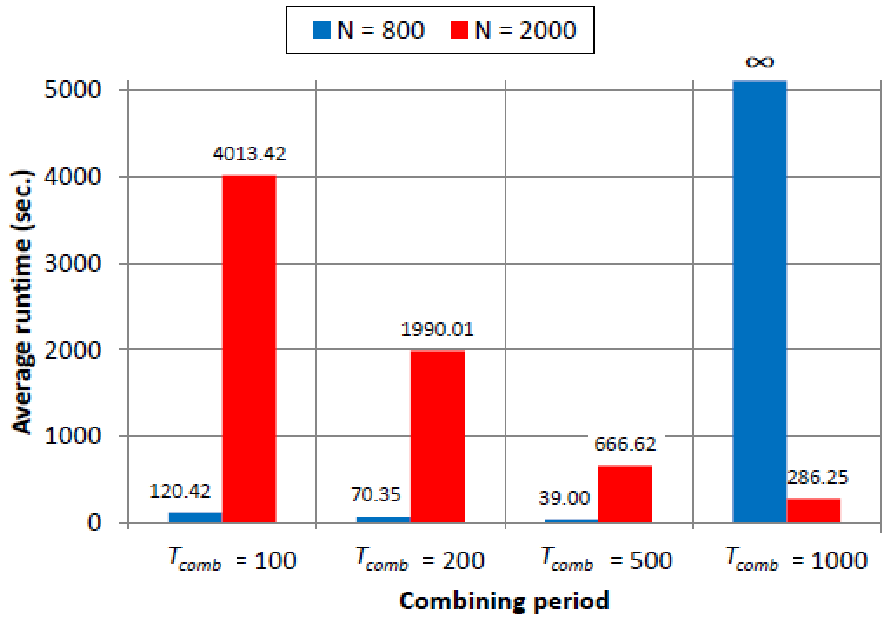


Figure 5.23: Execution time for binary MP20

Figure 5.23 depicts a comparison between a system with $N = 800$ in the blue bars and another having $N = 2000$ in red, facing a total of 200 000 trials. The former shows an incapability of solving the given case when T_{comb} is set with a great value, e. g., 1000. Such setting produces a failure in learning, making XCS-RC unable to tackle the given assignment.

For $T_{comb} = 100$, XCS-RC only requires 120.42 seconds to finish each run using $N = 800$. Contrary to that, 4013.42 seconds are used by the system with $N = 2000$ to provide similar performance, which is over 33 times as long. Then, by setting the combining period into 200, the system with $N = 800$ finishes each simulation in 70.35 seconds. This is very efficient compared to 1990.01 seconds required by the XCS-RC with $N = 2000$. The third comparison involving $T_{comb} = 500$ depicts similar results with 39.00 versus 666.62 seconds, won by XCS-RC with $N = 800$. Finally, no comparison can be made to systems having $T_{comb} = 1000$ since the corresponding XCS-RC fails to learn the task properly. This is an indication that by setting $N = 2000$, the system requires a relatively greater T_{comb}value.

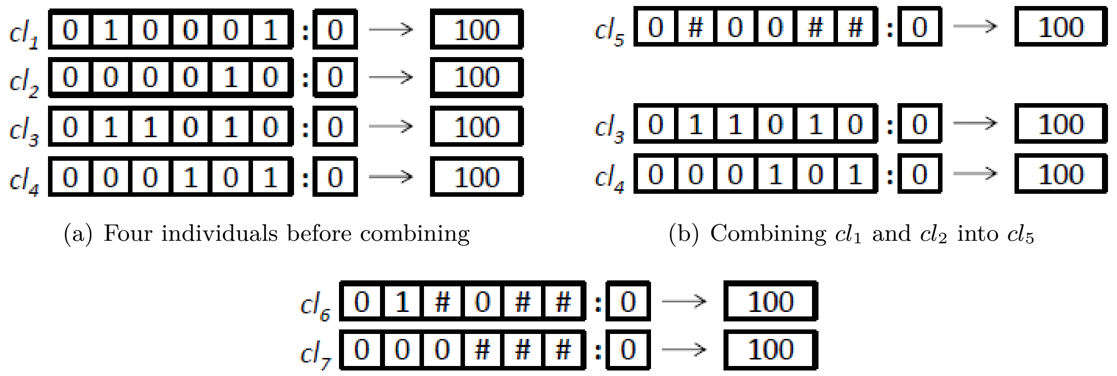
The results also show that in terms of execution time, simpler problems are more suitable to be handled with a lower value of N. However, as indicated by the results of the experiments, such setting potentially brings a drawback regarding the hastiness rate. Hence, it is advisable to set the parameters according to the characteristics of the problem. A complex problem is commonly indicated by a large input space, a low generalization possibility and a high variation of the reward map.

5.3.3 Optimal Population

A classifier system can only perform a high learning rate when its population contains a number of accurate rules. However, for environments with a large state space, the number of rules can be very huge. This makes the capability of performing generalization becomes very important, since each wildcard should be placed at an appropriate position. When all classifiers in the population are accurate and general, the system is guaranteed to behave as expected. An optimal performance will be achieved when the knowledge is represented by $[P]_{opt}$.

Simple cases like multiplexers are easy to analyze. Hence, determining the composition of rules that builds $[P]_{opt}$ is not difficult. However, XCS-RC fails to produce such a compact population in most of the simulations. The failure can be considered normal for real valued MP6 due to the large input space. Unfortunately, the proposed system is also unable to reach $[P]_{opt}$ in most of the binary MP6 assignments.

Figure 5.24(a) illustrates a set of classifiers in a population which is about to be generalized. The rules involved here follow the actual situation occurs in one of the conducted binary MP6 experiments. Now, as explained in Chapter 3, XCS-RC runs the parent-picking mechanism by prioritizing those with lower resemblance. A list consisting of the corresponding resemblance level is provided in Table 5.5.



cl1 0 1 0 0 0 1 : 0 → 100
cl2 0 0 0 0 1 0 : 0 → 100
cl3 0 1 1 0 1 0 : 0 → 100
cl4 0 0 0 1 0 1 : 0 → 100
cl5 0 # 0 0 # # : 0 → 100
cl3 0 1 1 0 1 0 : 0 → 100
cl4 0 0 0 1 0 1 : 0 → 100
cl6 0 1 # 0 # # : 0 → 100
cl7 0 0 0 # # # : 0 → 100

(a) Four individuals before combining

(b) Combining cl_1 and cl_2 into cl_5

(c) Combining cl_1-cl_3 into cl_6 and cl_2-cl_4 into cl_7

Figure 5.24: Combining results based on parent picking

In practice, the system checks one pair of classifiers at a time by prioritizing those with a lower resemblance level. When two or more pairs with identical resemblance level exist (e. g., cl_1-cl_2, cl_1-cl_3, cl_1-cl_2), the system picks a pair that involves a rule having the lowest index. This means, the pair cl_1-cl_2 becomes the priority. An absence of disproval allows the child (cl_5) to be accepted after eliminating the parents (see Figure 5.24(b)). Hence, by combining cl_1 and cl_2, the number of rules is reduced to three instead of previously four.

Table 5.5: Resemblance level for Figure 5.24(a)

No.	Resemblance	Pair
1	1	cl_3-cl_4
2	3	cl_1-cl_2
3	3	cl_1-cl_3
4	3	cl_2-cl_4
5	4	cl_1-cl_4
6	4	cl_2-cl_3

A more desirable solution is depicted by Figure 5.24(c) where the number of rules is reduced to two instead of three. Such result can be obtained if the system prioritizes the pair cl_1-cl_3 instead of cl_1-cl_2. However, since XCS-RC processes the rules using lower index priority, such thing does not commonly occur.

Despite a relatively satisfying performance, the proposed system shows a series of failures in reaching $[P]_{opt}$. The impact of such inefficiency is potentially greater for cases with a higher complexity. From the conducted investigations, some techniques may be implemented to overcome the problem, e. g., keeping the parents after the child is accepted. However, such solution potentially costs a significant negative impact to some other aspects of the system, especially regarding efficiency. Finally, a tolerance to this small drawback is picked over any potential inefficiency, after considering the desirable behavior shown by the system.

In the following chapter, an example of XCS-RC's possible development is described in the direction of performing a *Feature Selection* operation. Such functionality can only be implemented to a classifier system having a convincing generalization capability, denoted by the accuracy in placing wildcards. Using XCS-RC, such direction of development becomes possible, thanks to the implementation of the induction concept.

CHAPTER 6 FEATURE SELECTION

"Nature knows no pause in progress and development, and attaches her curse on all inaction."

(Johann Wolfgang von Goethe)

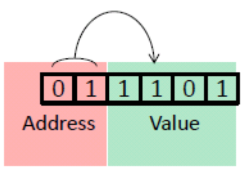
A capability of handling binary and real numbers can be considered as a basic feature of a classifier system. In facing those problems, the proposed method successfully shows a powerful and efficient performance compared to some versions which have been published previously, for both the learning result and the reduction of the space requirements. Naturally, such success is followed by a question: *what next?* Are there other capabilities to be developed after employing induction? And so this chapter is provided in responding to such line of questioning.

In performing learning, machines have some inherent constraints with respect to their resources, e. g., number of sensors and memory space. At the other hand, they are assigned to assist humans by coping with various tasks. Unfortunately, in the design phase, it is not always possible to determine beforehand which of the available information will actually be useful for decision making at runtime. Therefore, in addition to learning the assigned task, a system is also expected to identify the relevance of the available features. By doing that, the system may dynamically adjust its focus of attention only to the relevant information, thus reducing the usage of resources.

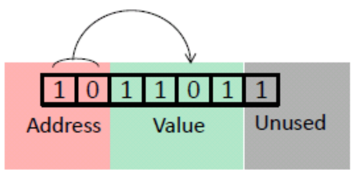
6.1 Relevance Level Identification

XCS-RC's capability of adequately placing wildcards opens up a number of development chances, e. g., a *Feature Selection* operation. Here, the system is expected to deliver another form of efficiency: filtering irrelevant environmental information. By receiving a series of information at each learning cycle, the proposed system attempts to identify the relevance level owned by each input element. A success in this attempt allows XCS-RC to optimize the space usage, and in turn, to make decisions involving physical entities such as turning off sensors.

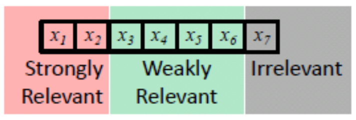
In [16], John et al., suggest a concept regarding two degrees of relevance, i. e., strong and weak. The former is used to express indispensable features while the latter implies a lower contribution to the system performance. Features that belong to neither of both are considered irrelevant, meaning that they can be completely disobeyed. A good example for describing relevance levels can be provided using an input for a 6-bit multiplexer.



(a) A normal MP6 input



(b) An MP6 input with an additional bit



(c) Relevance levels

Figure 6.1: Relevance in binary MP6 scenario

Figure 6.1(a) shows a "normal" input for binary MP6, consisting of two address and four value bits. Then, a different form of input is illustrated by Figure 6.1(a), with a total of seven input bits. Since the system only requires six elements at most, the last bit (i. e., the seventh) is completely unused. Therefore, as seen in Figure 6.1(c), it is considered irrelevant. Meanwhile, elements containing address information have a strong relevance due to their indispensable roles. Then, each of the value bits can be regarded as weakly relevant, since their values are not always decisive.

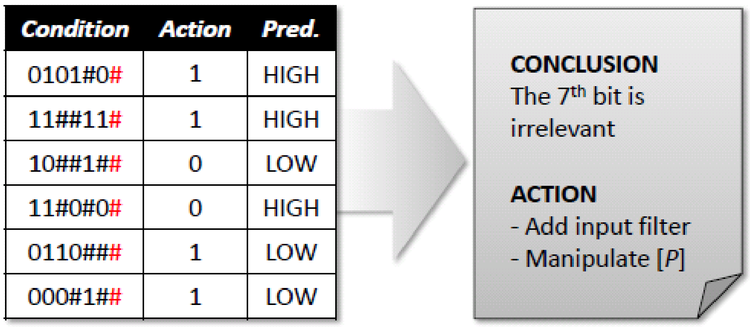
The feature selection functionality is added to XCS-RC in the expectation that the system will be able to detect irrelevant elements within the input structure. To accomplish such task, the system should be able to place wildcards correctly. In other words, a strongly relevant input element must not be replaced by a wildcard, while an irrelevant bit can always be. Therefore, performing such feature selection operation to the original XCS is not advisable, since it does not perform a convincing generalization capability.

Figure 6.2(a) illustrates a relevance level identification process. After a number of learning cycles, XCS-RC collects some knowledge in form of rules. At some point, the population is analyzed and a pattern is found: all the $7th$ bit of the conditions are replaced by wildcards. This matter denotes that the particular bit has a zero impact on the system decision. Therefore, it can be concluded that the last bit is irrelevant. XCS-RC responds to such conclusion with two actions, i. e., manipulating the classifiers in the population and adding a filter accordingly.

Those two actions are followed by manipulation of the condition attributes owned by all classifiers in $[P]$. As seen in Figure 6.2(b), all rules now have only six bits on their conditions instead of seven. Then, in order to adjust incoming inputs, Figure 6.2(c) depicts an addition of a filtering functionality. It intercepts all incoming inputs and eliminates the irrelevant bit. So, an environmental state "1010011" is filtered and the system processes it as "101001" instead.

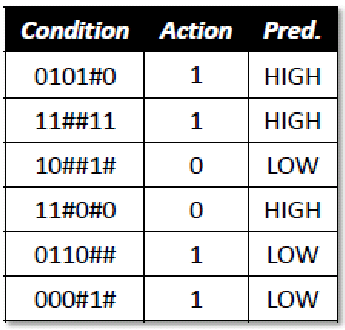
The provided illustration implies that a higher number of wildcards in a particular bit position is a sign that the bit is more likely to be an irrelevant one. Different to the unused element, the value bits are considered weakly relevant since some of their values should be kept specific. Finally, the address bits are strongly relevant, denoted by the absence of wildcards in all classifiers.

Filtering an MP6 input does not represent a complex feature selection problem. Nevertheless, the idea of a classifier system with a capability of identifying irrelevant elements is a desirable direction of development. Referring back to the architecture of an Organic



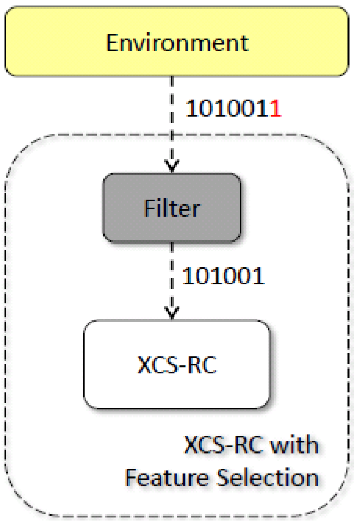
Condition	Action	Pred.
0101#0#	1	HIGH
11##11#	1	HIGH
10##1##	0	LOW
11#0#0#	0	HIGH
0110###	1	LOW
000#1##	1	LOW

(a) Conclusion regarding irrelevance



Condition	Action	Pred.
0101#0	1	HIGH
11##11	1	HIGH
10##1#	0	LOW
11#0#0	0	HIGH
0110##	1	LOW
000#1#	1	LOW

(b) Filtered population



(c) A modified XCS-RC

Figure 6.2: Feature Selection in XCS-RC

Computing system introduced in Section 1.2, such mechanism can be interpreted as a way of selecting an observation model (see Figure 1.6(a)).

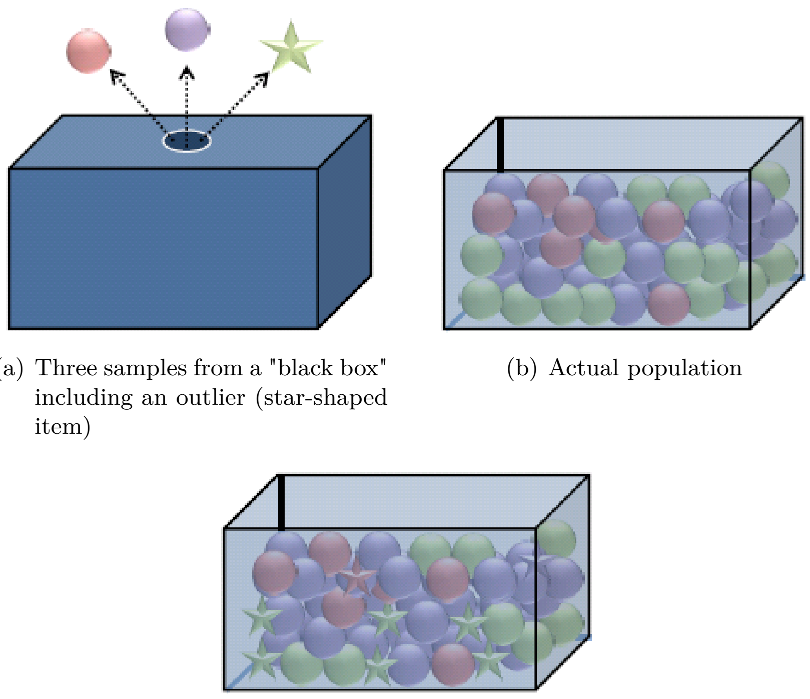
To implement such technique, a user-defined parameter T_{fil} determines the period for relevance checking mechanism. XCS-RC runs the feature selection operation every $T_{comb} \times T_{fil}$ trials, and starts the process by observing the number of wildcards at each bit position i. Then, the result is stored to an array $n_{\#}[i]$. A variable $\tau[i]$ keeps the relevance degree of the ith bit, where the value of each element is calculated following the formula $n_{\#}[i] \div n_{exp}$, where n_{exp} denotes the number of experienced rules in the population. Then, a user-defined relevance threshold parameter τ_{fil} whose value lays between 0 and 1, plays a role to determine the relevance degree of a specific bit position. Practically, the ith element of the conditions is considered irrelevant if the formula $\tau[i] \geq \tau_{fil}$ is satisfied.

Assigning a proper value for τ_{fil} is important, since it can be decisive in determining the success of feature selection operation. In simple environments like binary MP6, setting $\tau_{fil} = 1.00$ may suffice. However, a more complex environment commonly has a greater number of input elements. A larger input space urges the system to generalize a greater number of rules before identifying and filtering the irrelevant features out. Hence, for such systems, a lower relevance threshold value is generally more advisable.

Experiments show that the situation may get more difficult due to the appearance of the so-called *invalid disprovals*. Such term expresses low-experienced rules that disprove a great number of valid combining attempts. In performing feature selection, each of these individuals has similar effect to *outliers*. Some part of the obtained data that potentially drives the system to make incorrect conclusions. A detailed explanation regarding the issue is discussed as follows.

6.2 Outlier Detection

In performing feature selection, XCS-RC does not only require a number of new parameters. An additional mechanism in detecting invalid disprovals is also necessary, since they affect the system in a bad way. Such rules commonly have low accuracy, but tend to mislead the conclusion negatively. Despite their low experience, such classifiers disprove a significant number of combining processes which are actually appropriate. Figure 6.3 depicts the situation that illustrates how an outlier potentially lead the system to a false conclusion, using an alternate version of the black box example.


(a) Three samples from a "black box" including an outlier (star-shaped item)

(b) Actual population

(c) A mislead conclusion

Figure 6.3: Outlier effect

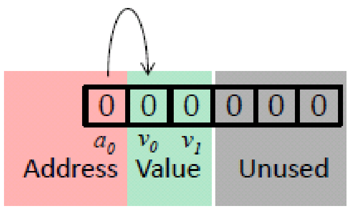
Figure 6.3(a) begins the illustration by showing three individuals taken from a black box, i. e., a red ball, a blue ball and a green star-shaped item. An observer is given a task to conclude the population of items in the box based on the discovered individuals. The actual content is shown in Figure 6.3(b), but another conclusion is more likely made, after taking the star into account (see Figure 6.3(c)).

XCS-RC makes a conclusion after considering experienced individuals in the population. When an offspring passes the disproval examination, the system concludes that the child is a valid combining result. Otherwise, an existence of one or more disproving classifiers causes the process to stop. Unfortunately, it is possible that a proper generalization is disproved by an inaccurate individual. In other words, the conclusion regarding a hasty combining result is mistakenly made, due to an influence of an outlier.

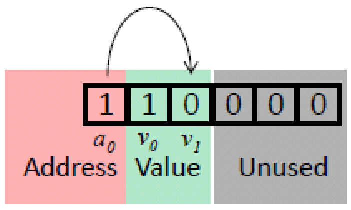
By the time such event occurs, the system still identifies the disproval as an accurate classifier. This is valid, since hastiness detection is only performed to rules that are currently in the action set. The situation where an inaccurate rule stays undetected does not commonly happen in "normal" cases. However, the chance becomes greater for scenarios involving a number of unused input elements. Therefore, an additional hastiness detection is implemented, taking a so-called *disproving rate* into account.

Figure 6.4 illustrates the role played by an outlying rule. Here, XCS-RC learns to perform a 3-bit multiplexer (MP3) that receives one address and two value bits. But instead of three elements, the environment sends a series of inputs in form of an array of six. The address is placed at the first bit (indexed by a_0), pointing one of the following two value bits (v_0 or v_1). The rest can be considered as irrelevant elements, having constant positions at the fourth to the sixth bit.

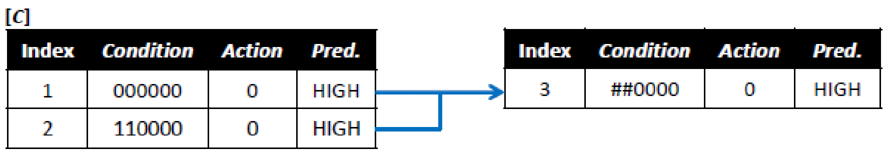
The aim of the task is not only to respond with the correct output, but also to identify the irrelevant bits within the given input. Such task is called "MP3+3", where the former "3" denotes the number of relevant bits and the latter refers to the irrelevant ones. As shown in Figure 6.4(a), an input "000000" is received, with an expected output "0". Some time afterwards, the system receives an input "110000", as depicted in Figure 6.4(b). Assuming that the system responds to both inputs correctly, the predictions of the corresponding classifiers are "HIGH".



(a) The address bit a_0 points to v_0



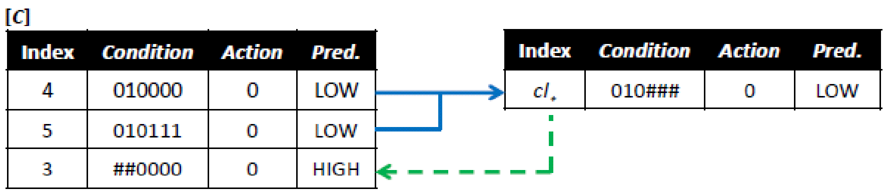
(b) a_0 points to v_1


[C]

Index	Condition	Action	Pred.
1	000000	0	HIGH
2	110000	0	HIGH

Index	Condition	Action	Pred.
3	##0000	0	HIGH

(c) Combined to a child


[C]

Index	Condition	Action	Pred.
4	010000	0	LOW
5	010111	0	LOW
3	##0000	0	HIGH

Index	Condition	Action	Pred.
cl_*	010###	0	LOW

(d) The child disproves a valid conclusion

Figure 6.4: Combining in MP3+3

Figure 6.4(c) depicts the situation that occurs when the system attempts to combine those rules. Wildcards are placed at the first two bits of the child's condition. The offspring, symbolized by "$\#\#0000 : 0 \rightarrow HIGH$", does not reflect a valid knowledge. Here, XCS-RC losses two out of three relevant information while all the irrelevant ones are kept. Finally, Figure 6.4(d) illustrates a situation when the child becomes a disproval for another combining process. Although the provided generalization attempt is actually valid, the system abandons the child candidate after being disproved inappropriately.

To solve this issue, XCS-RC monitors the disproving rate owned by each rule. This new attribute is calculated following a formula $cl.\rho_{disp} = cl.disp \div cl.exp$, where $cl.disp$ denotes how many times the rule disproves a combining attempt. A greater value of $cl_i.\rho_{disp}$ denotes the more likeliness of cl_i being an outlier.

Then, the decision to eliminate cl_i depends on the current number of experienced rule in the population (n_{exp}). If $cl_i.\rho_{disp} \geq n_{exp}$, then cl_i is removed from the population. The variable n_{exp} is picked as a threshold for deletion due to the fact that smaller population has less chance of combining. In other words, when the chance of being disproval is low, a high disproving rate raises an obvious suspicion.

To validate the concepts of feature selection and outlier detection, the system is tested using some specifically designed problems. Firstly, a multiplexer task is provided with a series of inputs having shuffled bits, plus some additional irrelevant elements. Secondly, an artificial scenario is used, involving a team of four predators with the task of capturing a prey. Three different learning systems is tested to cope with those tasks, i. e., the original XCS-RC, XCS-RC with feature selection (will be mentioned as "XCS-RC1" afterwards) and XCS-RC with feature selection and outlier deletion mechanism called "XCS-RC2". A detailed explanation regarding the experiments is provided as follows.

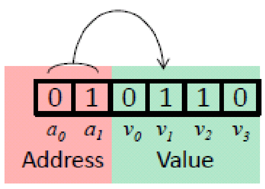
6.3 Multiplexers with Modified Input

In the first test, XCS-RC operates as a multiplexer receiving a series of 6-bit inputs with a modified composition. In a normal MP6 input, the first and the second elements contain an address while the rest are the value bits. Here, the positions are shuffled. This means, that each of the address bits may be placed in any of the possible six. To make the case solvable, the sequence is unchanged for the whole simulation. Figure 6.5(a) depicts an example of a normal MP6 input, while Figure 6.5(b) shows a shuffled version of that.

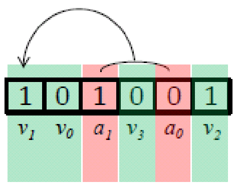
A normal MP6 input "010110" keeps the correct answer in the fourth bit (indexed with v_1). Here, v refers to the word "value" while the number "1" comes from the decimal representation of the address "01". Shuffling the elements means changing the positions of the bits randomly, but still keeping the correct answer at v_1. This way, the system has the chance to identify the new locations of all the elements and to act properly.

Such shuffled input cannot represent a feature selection task yet. Therefore, a number of bits are added to the array. As shown in Figure 6.5(c), 12 additional bits accompany the original six. Along the simulations, the positions of all address, value and unused bits are static. Then, XCS-RC is assigned to identify all of those irrelevant elements, along with the main goal of performing full correctness rate. This multiplexer scenario with six relevant and 12 irrelevant input elements is called "MP6+12".

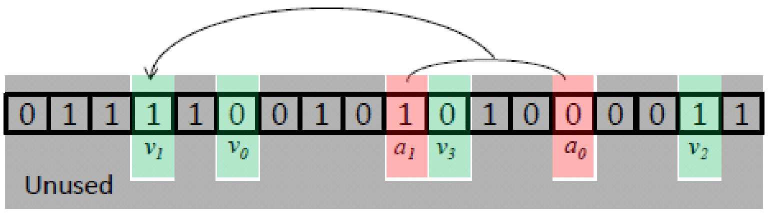
In this MP6+12 scenario, each suitable action is awarded by "1000" and otherwise "0". The investigation is conducted by feeding XCS-RC 20 000 binary inputs, and the decisions are taken using alternating modes between exploration and exploitation. The values for



(a) A normal input for MP6



(b) A modified MP6 input



(c) A modified 6-bit input with 12 irrelevant information (MP6+12)

Figure 6.5: Modification of MP6 input

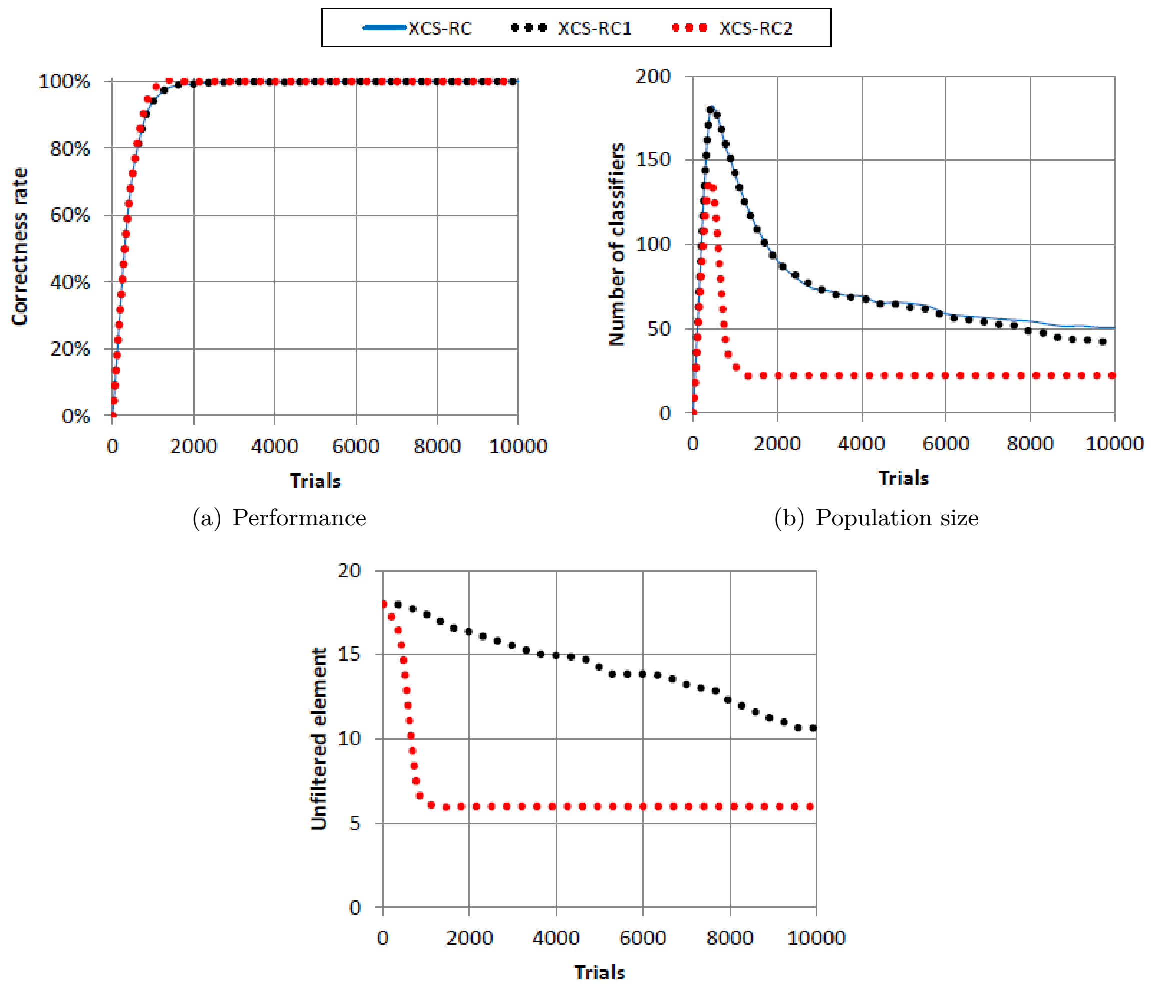
the feature selection parameters T_{fil} and τ_{fil} are 2 and 0.9, respectively. This means, that detection of irrelevant features is executed every second RC attempt, and a bit in a particular position is considered irrelevant when the amount of wildcards reaches 90% from the total experienced rules in the population. Other settings are set similar to that used in the MP6 task described in Chapter 5. All results provided here are averaged from 20 runs.

Figure 6.6(a) presents the correctness rate of all systems, where an achievement at 100% means that all inputs are responded correctly in the last 200 exploit trials. Here, each XCS-RC variant performs well and provides a stable performance above 95% after 1200 cycles. However, Figure 6.6(b) shows the superiority of XCS-RC2 (with filtering and outlier detection). The size in the population is quickly reduced to less than 30 individuals after 1200 trials. Having the same number of cycles, the others still cope with around 125 rules and never reach below 30 until the end of the simulations.

To analyze the capability of identifying irrelevant information, Figure 6.6(c) provides the comparison between XCS-RC1 and XCS-RC2. Here, XCS-RC2 successfully discovers all irrelevant bits after 1200 learning cycles. Contrary to that, XCS-RC1 only manages to filter nine out of 12 unused elements at the end of the simulations. This result clearly indicates that the outlier detection mechanism improves the capability of identifying irrelevant features.

It can be concluded that XCS-RC2 behaves more desirable to tackle the MP6+12 scenario in all aspects. The system shows a comparable level of performance along with a quicker reduction of the population size and a successful filtering attempt. To validate such conclusion, the learning systems are assigned to handle some more difficult tasks, i. e., the MP6+24 and MP11+11 problems.

The obtained results strengthen the previous conclusion that XCS-RC2 performs better than the others. Starting the comparison by discussing the capability of identifying irrelevant features, XCS-RC2 detects all unused bits after 2000 trials (see Figure 6.7(c)). XCS-RC1 shows a different behavior and never manages to perform the feature selection well, leaving 21 irrelevant bits unfiltered at the end of the runs. Another failure is shown by XCS-RC1 when attempting to reduce the population size. Figure 6.7(b) shows that the system ends



(a) Performance (b) Population size

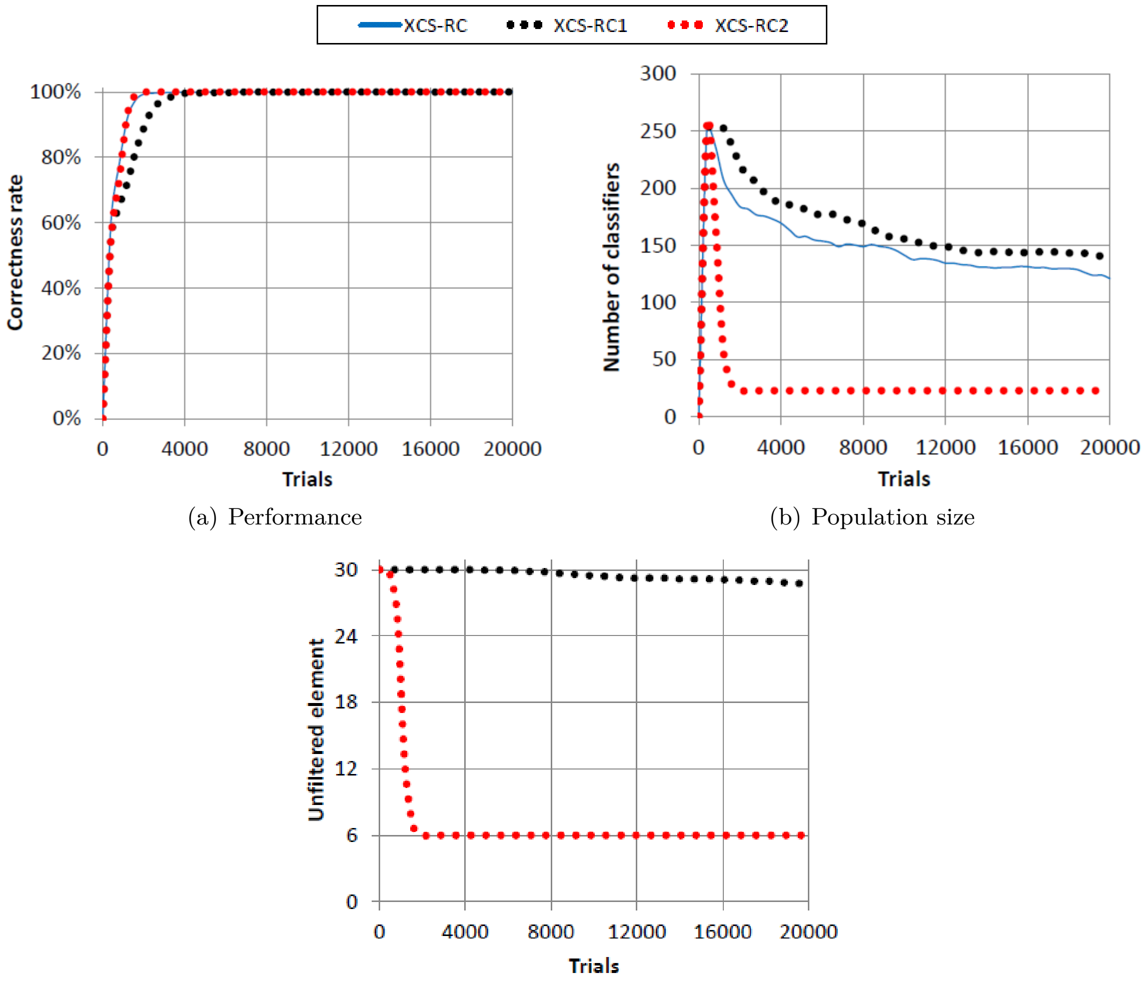
(c) Input length

Figure 6.6: Results for MP6+12

the simulations while still maintaining 138 rules, which is significantly higher compared to less than 24 individuals owned by XCS-RC2 since the 2000*th* cycle. With respect to the correctness rate, all three XCS-RC variants perform a similar level, as depicted in Figure 6.7(a).

For validating the capability of the proposed system in performing feature selection for multiplexers, a MP11+11 problem is given. Figure 6.8(a) shows that XCS-RC and XCS-RC2 perform at a similar level, responding to all inputs correctly after 8000 trials. Regarding space efficiency, XCS-RC2 manages to reduce the population size to below 40 rules (see Figure 6.8(b)). It is also shown that XCS-RC1 fails to perform a comparable behavior, providing a slightly slower learning speed and a significantly greater population size. At the end of the simulations, XCS-RC1 is only able to reduce the number of classifiers to around 265 individuals.

The conclusion that XCS-RC2 is superior to the others is finally validated by Figure 6.8(c) where XCS-RC2 requires only 4800 trials for filtering all 11 irrelevant elements out. Contrary to that, XCS-RC1 shows a poor behavior by only discovering one of the unused elements. The additional technique for detecting outliers can be considered useful to assist the system



(a) Performance

(b) Population size

(c) Input length

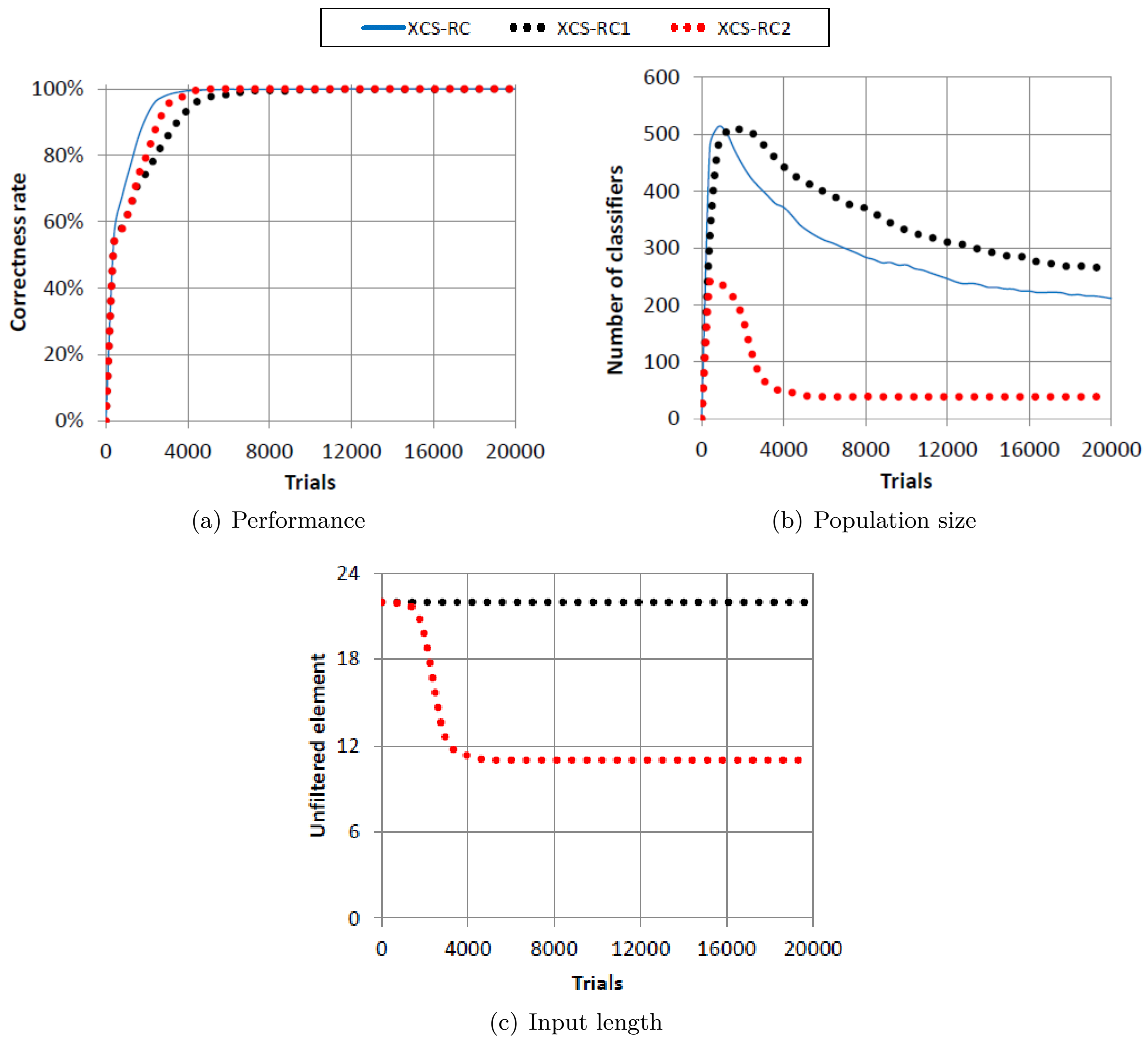
Figure 6.7: Results for MP6+24

in reaching the feature selection goal. Finally, these results remark the end of the comparison for multiplexer tasks. Following, XCS-RC is assigned to a new task having a whole different environment, i. e., the so-called Predator-Prey Pursuit scenario.

6.4 The Predator-Prey Pursuit Scenario

A number of predators has an assignment to pursue a prey and capture it. This Predator-Prey problem is a well-known task in the area of multi-agent systems (as in [21], [13] and [9]). Figure 6.9(a) shows a team of four predators (symbolized by red circles) and a prey (yellow circle) in an artificial torus (borderless) cellular automata world.

As depicted by Figure 6.9(b), each predator has limited vicinity of its surrounding. Currently, the predator P_2 sees the prey. When the location of the prey is known, each of the predators learns to make decisions that is advantageous to itself. At the same time, they should consider other possibilities that potentially contribute positively to the team goal (i. e., capturing the prey). A capture is defined as a situation where the predators success-



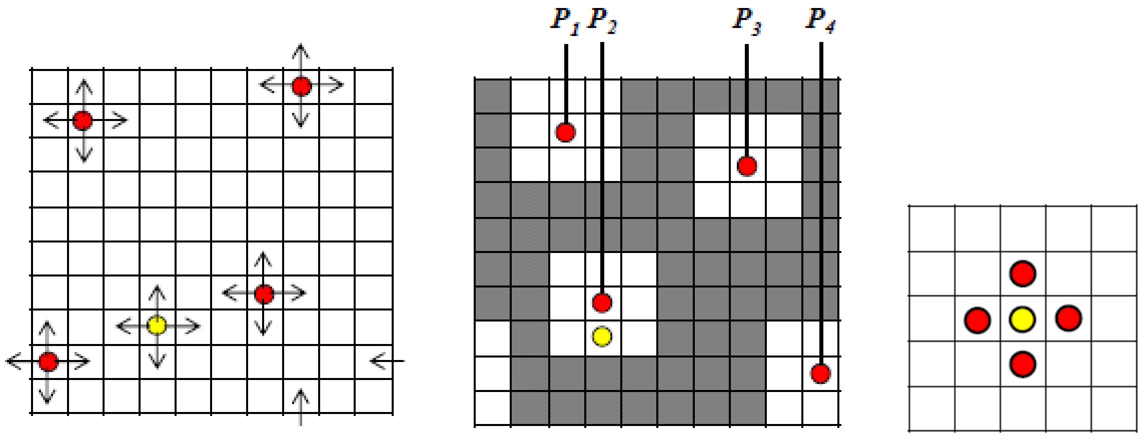
(a) Performance (b) Population size

(c) Input length

Figure 6.8: Results for MP11+11

fully surround the prey, occupying all four closest cells commonly known as the von Neumann neighborhood (see Figure 6.9(c)).

All entities begin the simulations at random coordinates. Each predator moves randomly when none of them sees the prey. When the prey's location is known by at least one team member, the coordinate is broadcasted. The information will be used by all predators, making each of them starts the learning process using their own population of knowledge. The system's response expresses a movement decision, meaning that each predator either moves north, south, east or west at every simulation tick. The option to not moving is unavailable. However, if the cell in the chosen direction is not empty, then the predator's position does not change. XCS-RC is assigned to assist the predators in learning to collaboratively capture the prey, as described in the following section.



(a) Four predators (red) and a prey (yellow)

(b) P_2 sees the prey

(c) The prey is captured

Figure 6.9: The predator-prey pursuit scenario

6.4.1 Learning

During the simulations, each predator decides its action autonomously. No external control is applied, and none of the teammates is capable of interfering any other's decision. Figure 6.10 depicts a flowchart of the predator's behavior at every simulation tick. After observing the surrounding, the predator checks whether the prey is seen or not. If the answer is no, it checks whether a message regarding the prey's coordinate is broadcasted from any of the teammates. If no such information is available, each predator simply moves to a random direction.

Once a predator obtains the prey's location (either directly seeing or receiving a broadcasted information), it translates the coordinate into direction. Taking an example provided in Figure 6.9(b), P_1 now knows that the prey is at the Southeast while P_2, P_3 and P_4 converts the information to the directions South, Southeast and Northeast, respectively. Now, the desired decision is to move towards the prey.

In [13], each predator performs a learning process by providing two information to XCS-RC, i. e., the prey's relative direction and the predator's local environment. Figure 6.11(a) shows an encoding process conducted by the predator P_2, which has a limited vicinity to 5×5 cells in the surrounding. As of this moment, it sees all the teammates and the prey, without any information regarding the cells depicted in gray color.

The first part of the input consists of four bits denoting the prey's relative direction, starting from North and continues clockwise. A value "1" means a confirmation that the prey is currently at the corresponding direction. Since P_2 sees the prey at west, the result of the encoding is "0001". Following a similar composition technique, P_1 translates the prey's location to "0011" meaning "SOUTHWEST". Then, P_3 encodes "NORTHWEST" to "1001", while P_4 obtains "1100" for "NORTHEAST".

For the second part representing the surrounding area, some options regarding limitation are available. Figure 6.11(b), 6.11(c) and 6.11(d) illustrate the types of neighborhood called von Neumann, Moore and extended Moore, respectively. The chosen observation range determines the number of input elements that will be fed to XCS-RC.

According to [13], the 4-bit von Neumann neighborhood is sufficient for the predators to

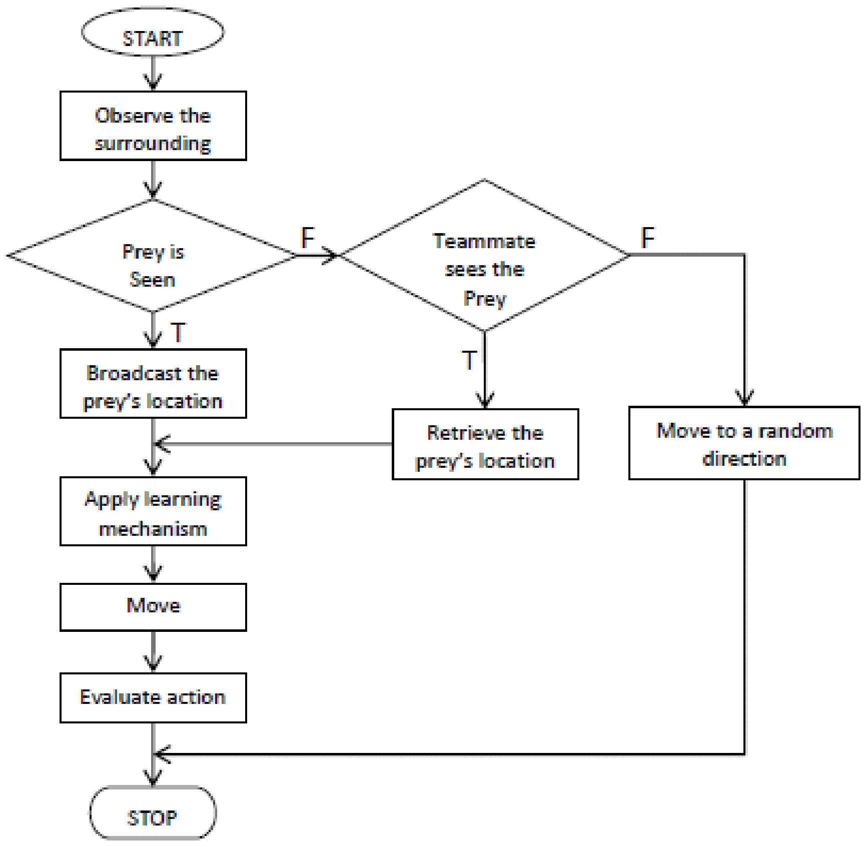
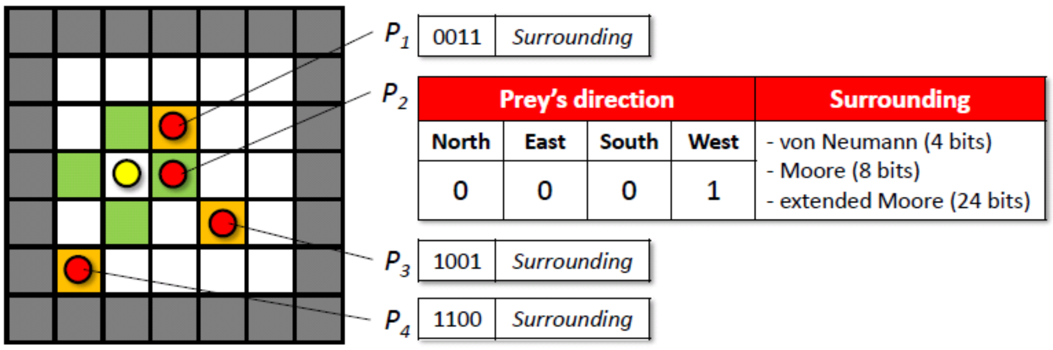


Figure 6.10: Flowchart of the predator's behavior

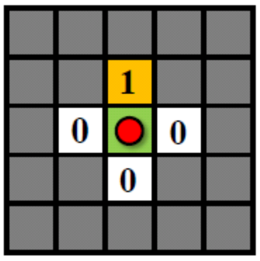
learn and to collaboratively pursuit the team goal. In composing this thesis, the others are also implemented for experimental purpose, i. e., to test the capability of XCS-RC with feature selection. By observing the Moore neighborhood, XCS-RC receives 12-bit inputs while the extended Moore gives the system an input consisting of 28 bits. A successful performance of feature selection is denoted by the capability of XCS-RC in reducing the number of input elements down to eight. This should be done without hurting neither the system performance nor the efficiency of space usage.

XCS-RC processes the inputs by performing the normal learning cycles. The reward "HIGH" is received if the last action leads the predator to getting *closer* to the prey, and "LOW" for *further*. A "VERY LOW" feedback is given when the distance does not change due to an attempt to move into a cell occupied by a teammate (*stagnant*). If the unchanged position is caused by trying to move to the prey's cell (implying a *closest possible* distance), then the reward is "HIGH". The last movement expresses a collaborative decision called *fair move*, and is rewarded "VERY HIGH". It can only occur when a predator moves from an advantageous position to let a teammate getting closer to the prey (see [13] for detailed explanation). Table 6.1 summarizes the five types of predator's movements along with the rewards.

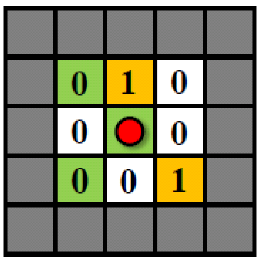
The assignment to guide the predators capturing the prey is more difficult than the multiplexer cases. Each predator has its own population of classifiers and learns separately without any social capabilities except broadcasting the prey's coordinate. Like in the previous tests



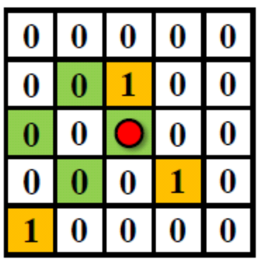
(a) Full composition



(b) Von Neumann neighborhood



(c) Moore neighborhood



(d) Extended Moore neighborhood

Figure 6.11: The input for XCS-RC

Table 6.1: Reward map for the Predator-Prey Pursuit scenario [13]

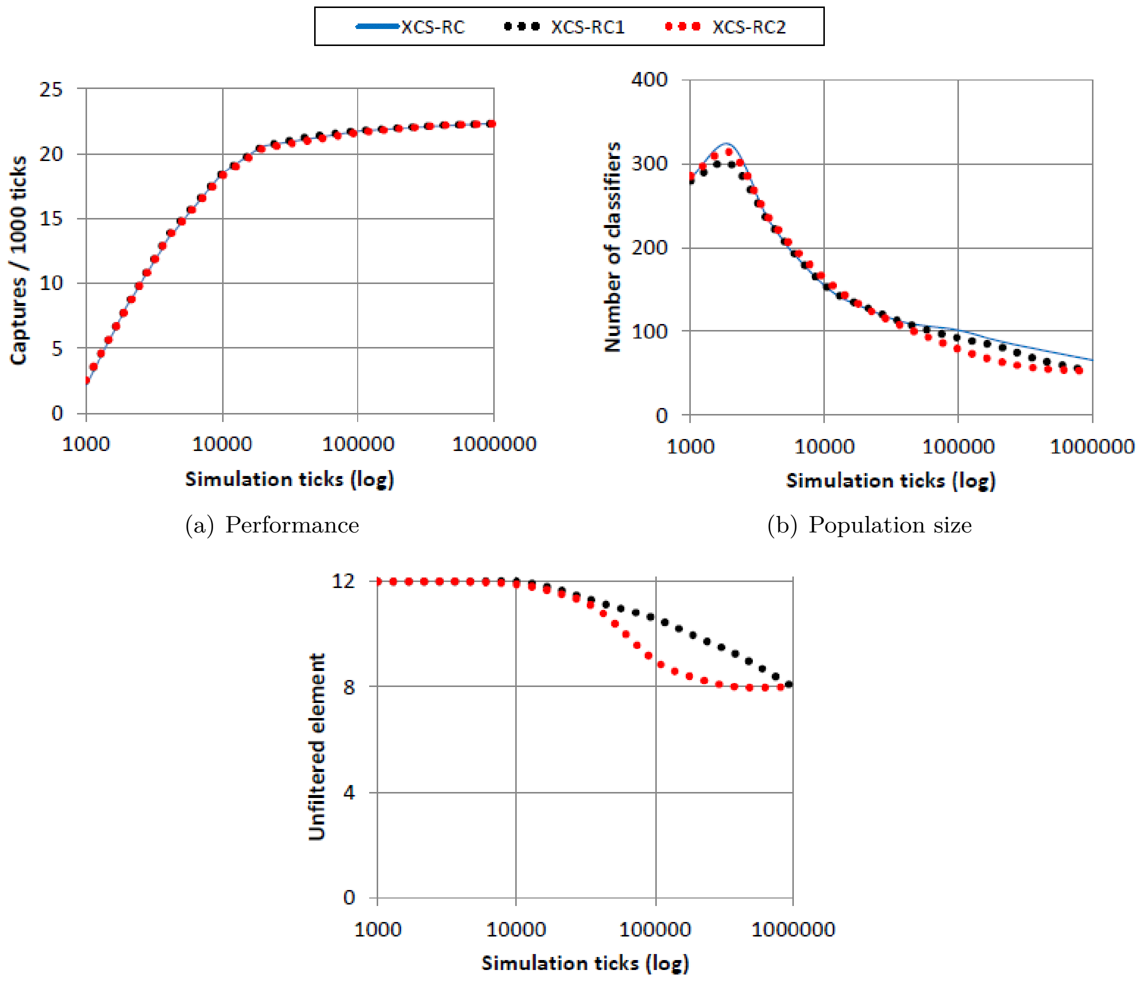
No.	Movement type	Reward
1.	Closer	HIGH
2.	Further	LOW
3.	Stagnant	VERY LOW
4.	Closest possible	HIGH
5.	Fair move	VERY HIGH

involving multiplexers, a series of investigations are run to compare XCS-RC, XCS-RC1 and XCS-RC2.

6.4.2 Results

For validating the impact of equipping the feature selection capability to XCS-RC, the predators start with some larger limitations, i. e., Moore and extended Moore neighborhood. It is expected that finally all of them succeed in reducing the number of observation elements. An identical set of system parameters and data acquisition technique are used in the simulations, unless stated otherwise. For each run, the length is limited to 1 000 000 simulation ticks. In every tick, each predator has the full autonomy to decide its movement. To provide a better impression on the comparison results, a static prey is chosen. This means, in the whole run, the prey does not move anywhere and only waits to be captured.

The figures here are presented quite differently to the previous graphics. In these results, the horizontal axes denote the number of simulation ticks in logarithmic scales. Then, the vertical axes indicate the average number of captures every 1000 ticks, accumulated from the beginning of the simulations, which should be maximized. All results are calculated based on an average of 10 runs, where the population size and the input length are obtained from all four predators, making it 40 sets of data in total.



(a) Performance

(b) Population size

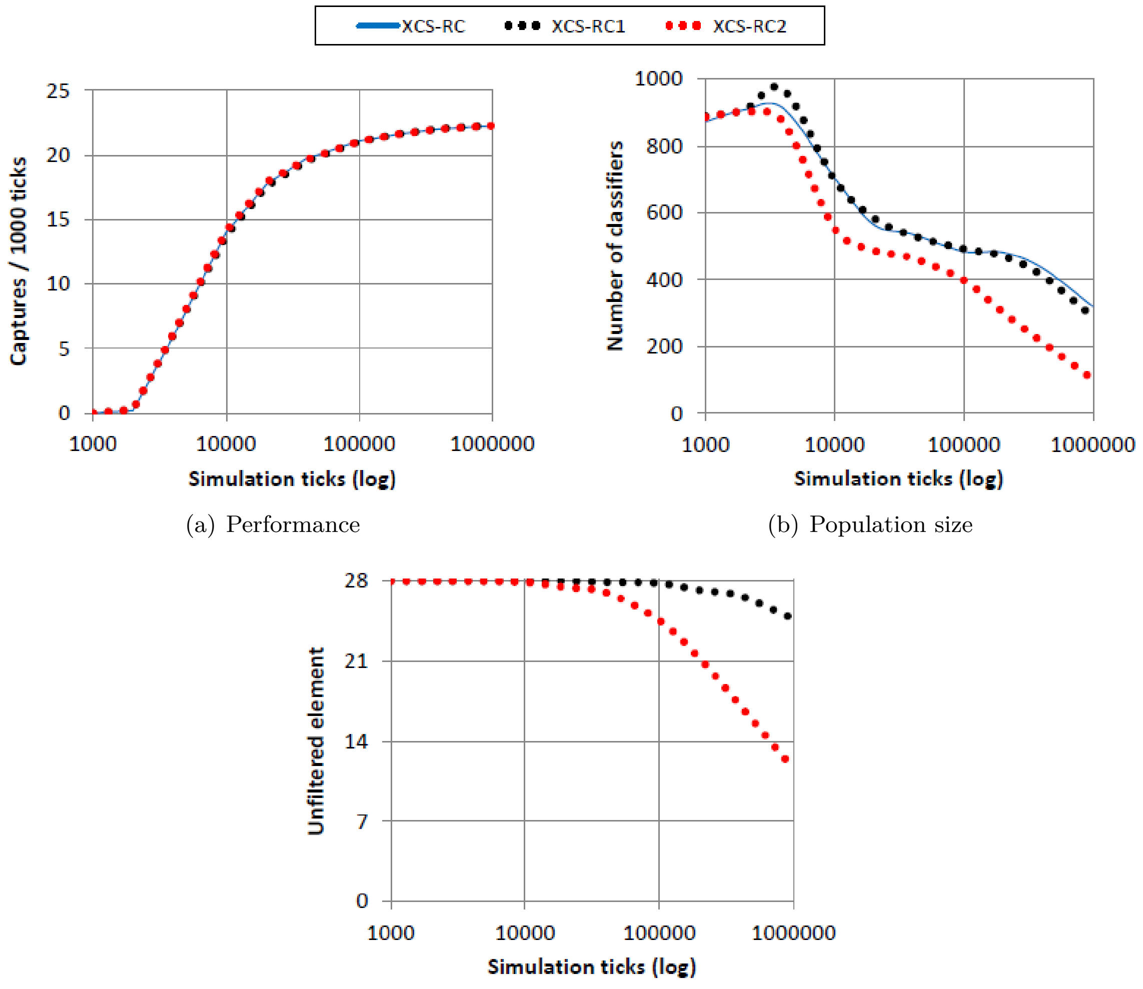
(c) Input length

Figure 6.12: Results for the Predator-Prey scenario with Moore neighborhood

Figure 6.12(a) depicts the performance of the predators in capturing the prey. Identical results owned by all learning systems indicate that the performance of XCS-RC is not affected by the additional feature selection and outlier detection techniques. Starting with zero knowledge, the team of predators ends up with an average of 22.3 captures every 1000 ticks, or approximately 22 300 for the whole run. Then, Figure 6.12(b) provides a similar indication concerning the generalization capability. In reducing the number of rules, XCS-RC, XCS-RC1 and XCS-RC2 show insignificant differences in the results.

However, a noticeable difference is shown in Figure 6.12(c) that compares the capability of XCS-RC1 and XCS-RC2 in minimizing the number of input bits. The latter, equipped with a mechanism to detect outliers, filters out irrelevant information quicker than the former. After 80 000 ticks, XCS-RC2 already eliminates two irrelevant bits while XCS-RC1 has not succeeded to remove any. The highest gap between them occurs around the 200 000*th* tick where XCS-RC1 provides an average at 10.375 bits while XCS-RC2 already copes with 8.8 elements. At the end of the simulations, XCS-RC2 slightly outperforms the counterpart by 8.075 to 8.25 elements within the inputs.

The superiority of XCS-RC2 continues to appear in the next task, assisting a team of predators using the 24-bit extended Moore neighborhood. Figure 6.13(b) illustrates how the systems manage to minimize the population size over the number of ticks. Here, XCS-RC2 outperforms the others, ending the runs with an average of less than 105 classifiers. Meanwhile, XCS-RC maintains around 320 rules and XCS-RC1 still copes with approximately 290 individuals at the end of the simulations. This result shows the indication that XCS-RC2 has the quickest reduction mechanism compared to the others.



(a) Performance

(b) Population size

(c) Input length

Figure 6.13: Results for the extended Moore neighborhood

In the attempt of filtering out irrelevant data, XCS-RC2 ends the comparison by providing a better result. As shown in Figure 6.13(c), the average of the input lengths are 11.5 bits for XCS-RC2 and 24.7 elements for XCS-RC1 at the end of the simulations (1 000 0000 ticks).

Finally, Figure 6.13(a) confirms that the extra capabilities owned by XCS-RC1 and XCS-RC2 do not affect their performance negatively. The conclusion is made based on the fact that the number of captures is very similar for all classifier systems in the whole simulations.

These results provide a set of evidences showing that a feature selection capability can be implemented to XCS-RC without bringing any negative impact. Moreover, it even manages to make the learning process more efficient and desirable. The development of XCS-RC in the direction of feature selection is promising, leading to future investigations. This chapter successfully provides a convincing argument regarding an instance of future developments. Finally, some concluding remarks and outlooks close the discussions in this thesis, as presented in the following chapter.

CHAPTER 7 CONCLUSION AND OUTLOOK

"And whoever today is better than yesterday, they are the lucky ones."

(Muhammad)

Improvement is a keyword that describes the motivation behind the proposed method. The composition of XCS-RC has a specific aim, i. e., to gain a more powerful learning system that is able to deliver better results compared to the original XCS and its variants. Therefore, the proposed system is designed based on a philosophical approach that has been conducted by humans for centuries. Inductive reasoning, the technique that founds the whole investigation, is a classical method invented by Aristotle thousands of years ago.

Since then, the induction concept has been developed widely. Notable names, e. g., Tycho Brahe, Nicolaus Copernicus, Alhazen, Galileo Galilei, Isaac Newton and Charles Darwin, have contributed to improve the stability of the theory. Induction became the basis of scientific activities where humans collect their experiences and build their knowledge. They all conduct the same things, observing the environment, obtaining a set of data, followed by the composition of conclusions based on some underlying patterns. Therefore, the quality of the conclusions highly depends on the accuracy of every individual datum. Not only that, the comprehensiveness of the data plays an important role to prevent a hasty generalization.

A series of investigations is conducted in the attempt of adapting such method to machine learning, taking an example using the Accuracy-based Learning Classifier System (XCS). In building the knowledge, XCS employs reinforcement method to perform online learning. The result is stored in form of classifiers which contain some attributes expressing the covered states, the advocated action and the suitability between both of them.

After several years of development, XCS has been established as one of the most successful technique in the genetic-based machine learning area. Its capability of handling various forms of input is one of the positive aspects. Moreover, the classifier uses a coded language that can be easily translated to a simple human language. Such factors are very helpful for the users, especially for evaluation purposes.

However, the performance of XCS does not always satisfy the common needs. The fact that the system learns quite slowly even for some simple tasks, is a flaw that requires to be solved. And so this thesis is composed, attempting to contribute to the scientific world by improving the learning method for classifier systems. The modifications mainly affect a specific component with an assignment of discovering general and accurate rules. Classifiers having such positive characteristics increase both the effectiveness and the efficiency of the learning process, and the results can be significantly improved.

7.1 Summary

In this thesis, the induction concept is derived to perform a *heuristic-based genetic operation* called *Rule Combining*. The system, namely *XCS with Rule Combining* (XCS-RC), is designed to make conclusions based on the experience. After observing the learned knowledge, generalized offsprings are created, replacing the old individuals.

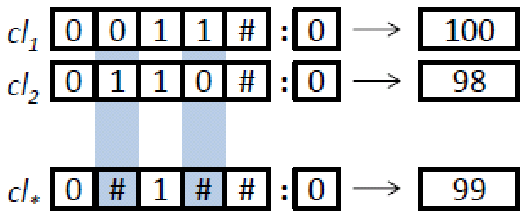

cl_1 0 0 1 1 # : 0 → 100
cl_2 0 1 1 0 # : 0 → 98
cl_* 0 # 1 # # : 0 → 99

Figure 7.1: Combining

The whole modification is based on the induction concept illustrated in Figure 7.1, stating that *when a "0" meets a "1", they are combined and become "#"*. However, to become a parent, a rule should be sufficiently experienced. Then, there are three basic principles that should be fulfilled before a combining process is executed, i. e., the parents advocate the same action, they have insignificant difference of predictions, and no disproving classifier exists in the population. In the implementation of the latter, every combining process should be approved by all rules representing the existing knowledge. The "communal acceptance" is followed by equipping the child with necessary attributes, and then insertion before finally removal of the old individuals.

Afterwards, the system prepares an anticipative mechanism due to the possibility that the child is actually created through a hasty conclusion. In practical, every executed rules are always monitored. Once a generalized classifier shows an intolerable behavior, it is removed and replaced by a new one. Such knowledge correction mechanism is proven useful in strengthening the reliability of the proposed system.

To minimize the possibility of making hasty generalization, XCS-RC conducts a simple experiment when picking an action. In the exploration mode where normally the result is taken from a random process, the system deterministically chooses an output having the lowest experience. By doing this, shorter time is required to obtain more knowledge. And by employing no wildcards in covering, the system guarantees that all ungeneralized rules are accurate.

Figure 7.2 illustrates the summary of modifications using a schematic diagram. Each rectangle represents an activity while the arrows with solid and dashed lines refer to steps and impacts, respectively. RC technique is affected by two modifications, i. e., ungeneralized covering mechanism and deterministic explorative decision. The impact particularly affects the parent picking process and the result of disproval examination. Then, the generalization result is responded by the knowledge correction mechanism. In turn, a deletion to a generalized rule makes the covering recover the missing states, which will affect the behavior of the explorative action selection. Finally, the whole modifications are implemented to the original

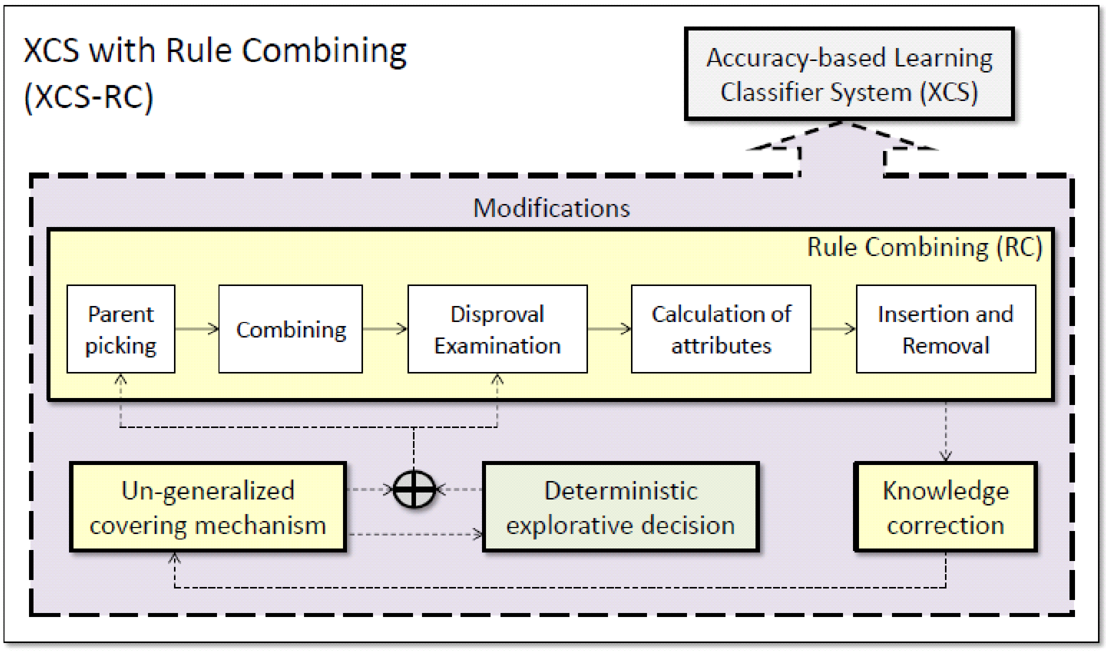

XCS with Rule Combining (XCS-RC)
Accuracy-based Learning Classifier System (XCS)
Modifications
Rule Combining (RC)
Parent picking
Combining
Disproval Examination
Calculation of attributes
Insertion and Removal
Un-generalized covering mechanism
Deterministic explorative decision
Knowledge correction

Figure 7.2: Schematic diagram of the system modifications

XCS, replacing some unnecessary mechanisms.

Such method has a direct impact to decrease the space requirement, but not limited to that. As a result, XCS-RC manages to present at least three significant improvements compared to the original XCS:

1. Quicker learning rate
2. More powerful generalization capability
3. Simpler parameter setting

Those changes are confirmed through a series of experiments involving various cases, i. e., multiplexers, checkerboards, and Markovian environments. Not only in handling binaries, but also in some experiments involving continuous values. Moreover, the proposed method also enables the system to perform a feature selection operation, an additional capability that is less likely to be applied to any other variants of XCS. More detailed explanation regarding the improvements presented by XCS-RC is discussed as follows.

7.1.1 Learning rate

The first aspect that determines the judgement of XCS-RC is the learning rate, which combines the factors of correctness rate and number of required cycles. Table 7.1 summarizes the achievement of XCS, XCSR and the proposed system presented in Chapter 5, for the single-step cases. Here, the comparison focuses on the stability of the performance, i. e., the number of required cycles to get a stability at 95% and 99% of the full performance. A 100% indicator is not used to tolerate insignificant changes.

Starting with binary multiplexers, the number of required trials indicates that XCS-RC performs a better learning rate compared to XCS. The proposed method requires significantly less trials to achieve a stable 95% of full correctness rate or above. Similar to that, both systems achieve a 99%-stability after a significant different number of learning cycles, where XCS-RC shows a more desirable behavior.

Table 7.1: Performance comparison for the single-step problems

Test scenario	Total trials	Stable at 95% or above (trials)		Stable at 99% or above (trials)	
		XCS/XCSR	XCS-RC	XCS/XCSR	XCS-RC
MP6 binary	10 000	1400	200	2600	300
MP11 binary	30 000	9300	1000	15 700	1400
MP20 binary	200 000	74 500	9000	88 000	12 000
MP6 real	40 000	20 600	16 600	*unknown*	*unknown*
MP6 real (alt.)	40 000	*unknown*	13 000	*unknown*	*unknown*
CB-2d3	50 000	*N/A*	1800	*N/A*	5000
CB-2d5	50 000	*N/A*	4400	*N/A*	13 600
CB-2d7	50 000	*N/A*	11 800	*N/A*	26 800
CB-3d3	50 000	*N/A*	16 000	*N/A*	34 600

Then, the results from the real MP6 simulations strengthen the indication of XCS-RC's superiority even more. For the task involving "normal" thresholds, XCS-RC manages to get a stable 95% of full corectness rate quicker than the counterpart, i. e., 16 000 against 20 600 cycles. However, due to the large input space, both of them fails to get a stability at 99% or above (denoted by the term "unknown"). In the problem involving alternating thresholds, the proposed system reaches a stability at 95% of full performance after 13 000 trials while XCSR fails to present a similar result. As previously discussed, such task is more difficult for XCSR since the environment contains a set of range having relatively narrow widths.

Despite the results unavailability for XCSR in the checkerboards ("N/A" refers to "Not Available"), XCS-RC proves its ability to map the given cases. Nevertheless, an assumption can be made based on the results from the real MP6 case. The checkerboard environments are more difficult to solve due to the existence of multiple thresholds in each dimension. Based on the fact that XCSR fails to solve real MP6 with alternating thresholds, one can conclude that the possibility of reaching a stability at 95% of full correctness rate is low. Unfortunately, no data can be provided to back such conclusion.

Table 7.2: Performance comparison for the multi-step cases

Test scenario	Total trials	Average steps	Tolerable value	Stability (trials)	
				XCS	XCS-RC
Woods1	8000	1.6875	1.8148	400	200
Woods2	8000	1.6875	1.8148	500	200
Maze4	8000	3.5000	3.8537	200	200
Maze5	8000	4.6111	5.0702	800	200
Maze6	8000	5.1944	5.6415	400	200

Finally, the problems involving Markovian environments are presented here to compare XCS and XCS-RC in facing multi-step tasks. Instead of using percentage indicators, another approach is used to identify the success of the systems. Each map has its own expected value depicting the average number of steps from all empty cells. A tolerance is applied based on the assumption that the animat starts from those further cells more often than the closer ones. Here, the term "further" refers to the cells requiring more steps than the average.

In finding the food within the provided environment, the animat is guided by the learning systems to minimize its steps. As seen in Table 7.2, both XCS and XCS-RC succeed to accomplish the task and perform a slightly different effectiveness. XCS-RC requires 200 trials to reach a tolerable value for mapping the given environments, while XCS reach similar value by performing 400 and 500 attempts for Woods1 and Woods2, respectively. For the Mazes, XCS-RC successfully does all jobs with no more than 200 trials while XCS requires a slightly more cycles to solve the given problems.

As a conclusion, XCS-RC is more superior than XCS in terms of learning rate. It manages to fulfill all the given tasks in comparable level or even better. Those effective processes depict its ability to manage an efficient population of general and accurate rules, as explained in the following subsection.

7.1.2 Generalization capability

It is guaranteed that a classifier system having a set of accurate rules would perform effectively. However, when those classifiers are specific (ungeneralized), the space demand is high, making it highly inefficient. As the main modification of the proposed system, RC replaces the role of XCS's Darwinian genetic operators in generalization tasks. By combining rules, the size of the population can be decreased. Along with some additional techniques, XCS-RC performs a more desirable behavior compared to the original system and the previously published variants.

Table 7.3: Population size comparison

Test scenario	Total trials	Size of $[P]_{opt}$ (rules)	Final $[P]$ size (rules)		Stability (trials)	
			XCS	XCS-RC	XCS	XCS-RC
MP6 binary	10 000	16	45.300	22.850	*unknown*	700
MP11 binary	30 000	32	112.250	48.600	*unknown*	6800
MP20 binary	200 000	64	344.850	149.350	*unknown*	*unknown*
MP6 real	40 000	16	664.800	93.050	*unknown*	*unknown*
MP6 real (alt.)	40 000	16	661.000	84.275	*unknown*	*unknown*
CB-2d3	50 000	18	*N/A*	19.450	*N/A*	*unknown*
CB-2d5	50 000	50	*N/A*	52.500	*N/A*	*unknown*
CB-2d7	50 000	98	*N/A*	164.850	*N/A*	*unknown*
CB-3d3	50 000	54	*N/A*	146.250	*N/A*	*unknown*
Woods1	8000	31	162.350	31.350	*unknown*	500
Woods2	8000	31	161.700	32.550	*unknown*	6800
Maze4	8000	103*	601.000	105.750	*unknown*	1900
Maze5	8000	183*	954.850	186.400	*unknown*	2400
Maze6	8000	175*	723.100	177.550	*unknown*	5100

**optimal result out of 100 simulations*

Population size is the second priority after learning rate. Nevertheless, the conducted experiments indicate that the system performance has a strong dependency to the obtained knowledge. The system learns better when the set of classifiers is able to represent the learned knowledge efficiently. Having a small population size does not always mean that the behavior will be better. But instead, the accuracy of the rules determines the learning quality. A proper generalization method leads to an optimal learning efficiency, and in turn, to a system with high effectiveness.

Table 7.3 contains a collection of experimental results, particularly regarding the population size. It is an interesting phenomenon that XCS and XCSR have a stable population in none of the experiments. Except for those involving checkerboards, the number of rules still fluctuates at the end of the simulations. Contrary to that, XCS-RC provides a stable population size in nine out of 14 tasks. Those unstable sets occur when facing problems having a considerably large input space, where more trials are commonly required.

When performing binary multiplexer, XCS-RC is not able to reach optimal population in any of the tasks. The proposed system collects a stable MP6 population with 22.85 rules since the $700th$ trial, which is relatively close to the optimum at 16. Since XCS still copes with 45.3 classifiers at the end of the simulations, XCS-RC can be considered as the winner. The results for the MP11 task also show similar indications. XCS-RC manages to reduce the number of rules down to 48.6, despite the optimal value at 32. At the end of the MP20 simulations, the population size owned by XCS-RC is 149.35 individuals, and still converging. Although it is unable to reach the ideal size at 64 classifiers, those results indicate a significant improvement compared to those provided by XCS that ends the runs with 344.85 rules.

In tasks involving real MP6, XCS-RC fails to manage a stable population. Although the ideal size for both normal and alternating thresholds is 16 rules, the proposed system only succeeds to reduce the value down to 93.05 and 84.275 classifiers, respectively. Nevertheless, this is better compared to XCSR which maintains a far higher amount, i. e., around 660 rules at the end of the simulations.

The proposed system is also unable to form $[P]_{opt}$ in all four checkerboard tasks. The cause of this failure is the small number of trials, which is limited to 30 000. Nevertheless, XCS-RC manages to reduce the number of rules down to 19.45, 52.5, 164.85 and 146.25 individuals for CB-2d3, CB-2d5, CB-2d7 and CB-3d3, respectively.

Finally, in experiments involving multi-step tasks, XCS-RC shows no significant difference of effectiveness compared to the original system. Hence, the winner is determined by the one having a lower population size. The reference value (size of $[P]_{opt}$) for Maze is taken by running a series of 100 simulations. The minimal value among the results is assumed as the optimum, after previously examining the validity of the contained classifiers.

XCS-RC forms a set of populations whose sizes very close to the target. All tasks are learned successfully, having no more than four classifiers greater compared to that owned by $[P]_{opt}$. In this matter, XCS fails to perform similarly. The original system maintains a much higher number of rules compared to XCS-RC. These facts end the discussion regarding the generalization capability with a strong impression, that the RC technique contributes strongly on making a compact population. The achievement is significantly better than that provided by the Darwinian genetic operators.

7.1.3 System parameters

Every learning system is equipped with a set of adjustable parameters for a flexibility purpose. The number of variables denotes the degree of freedom owned by the system. Some simpler problems may require different parameter setting to those which are more complex. For instance, XCS is able to solve Woods scenarios using a wildcard probability $P_{\#} = 0.33$, but the more suitable value for coping with Maze is 0.05. A series of experiments shows that the value above 0.1 may cause XCS to fail in performing a proper learning process.

However, a greater number of parameters does not necessarily indicate a better learning quality. As a matter of fact, compared to the original system, XCS-RC employs less user-defined variables. Table 7.4 summarizes the difference of both systems with respect to the parameters, including those required in performing the feature selection operation.

Table 7.4: Unused XCS parameters and the new ones for XCS-RC

No.	Unused XCS parameters	XCS-RC parameters
1	Wildcard covering probability ($P_{\#}$)	Combining period (T_{comb})
2	GO threshold (θ_{GA})	Minimal experience ($minExp$)
3	Crossover probability (pX)	Prediction tolerance ($predTol$)
4	Mutation probability (pM)	Prediction error tolerance ($predErrTol$)
5	GO prediction error reduction	Feature selection period (T_{fil})
6	GO fitness reduction	Relevance threshold (τ_{fil})
7	Minimal experience to subsume (θ_{sub})	-
8	Deletion threshold (θ_{del})	-

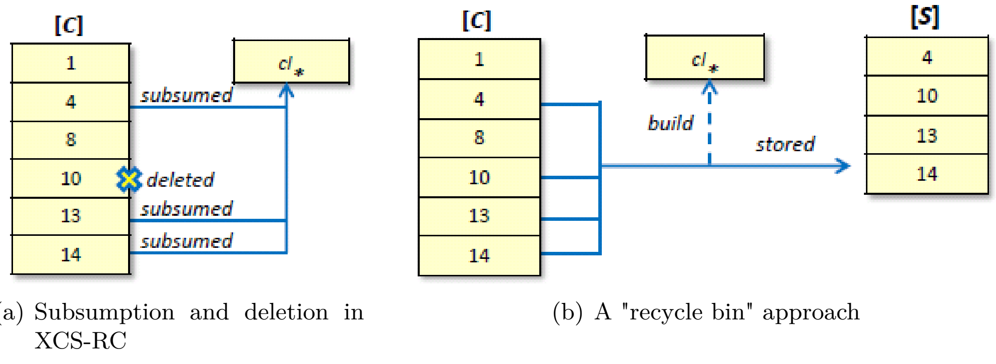
Eight obsolete parameters owned by XCS are related to the discovery component whose main task to produce general and accurate rules. Five of them are used in genetic operation, which is completely unused in XCS-RC. Instead, the proposed system entitles four new parameters for performing RC, plus additional two used in feature selection. Another important note is that there are three parameters owned by XCS that play direct roles in some random processes. Contrary to that, XCS-RC does not perform such randomizations and thereby employs no probability parameter.

XCS-RCS shows a high flexibility in solving the given assignment despite its a smaller amount of parameters. At the positive side, lower number of user-defined variables makes the system easier to be operated by human. Moreover, elimination of probability parameters contributes to a more accurate evaluation. An additional functionality of feature selection even improves the system efficiency, by equipping XCS-RC a capability to identify any irrelevant information.

7.2 Outlook

Despite those improvements, the conducted experiments provide some indications that XCS-RC owns some noticeable drawbacks. Additional approaches might be implemented to minimize the flaws, but it is difficult to apply them without facing some consequences, e. g., complicated calculations, process inefficiency. Therefore, those unavoidable impacts can be considered as a payoff to the huge advantage offered by the proposed system. To complete the discussion, this final section is dedicated to provide some outlook containing possible further developments.

Among issues that are worth mentioning, loss of knowledge is the most harmful occasion. As seen in Figure 7.3(a), a child is completed after subsuming experienced classifiers. Such removal is necessary for efficient usage of storage. However, an unexpected risk of losing knowledge might occur. Later when being executed, if the child is identified as a hasty product, it will be removed. This means, that the knowledge it subsumed, is also lost. Other tricks (e. g., temporarily keeping the old individuals) are possible, but there is a high risk of having an overpopulated storage, especially in cases involving a large input space.



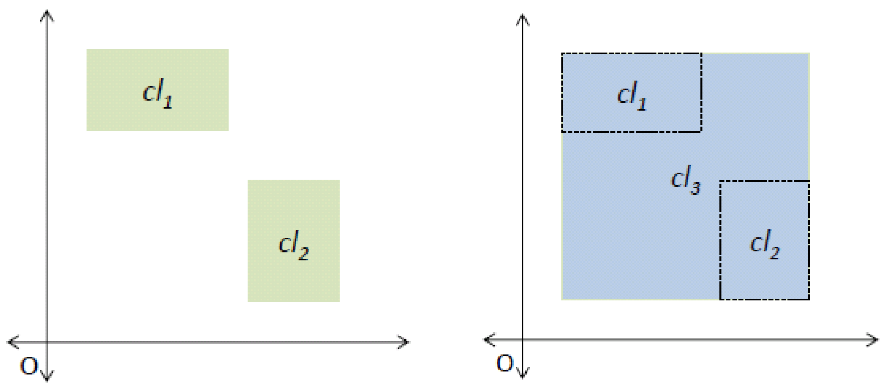
(a) Subsumption and deletion in XCS-RC

(b) A "recycle bin" approach

Figure 7.3: Alternative method for anticipating loss of knowledge

Another option to the problem is by implementing a "recycle bin" approach. In practical, the old individuals are stored to a so-called *Storage set*, as illustrated in Figure 7.3(b). This might solve the problem, but yet another consequence shows up. Although a child is always created based on two classifiers, it practically replaces a greater number of rules in many occasions. So, for scenarios with a large input space (e. g., binary MP20 or real MP6), this trick potentially causes a great inefficiency. To keep the system from such complexity, none of those options is implemented to XCS-RC.

Beside the flaw of learning inefficiency, the proposed system also fails to reach $[P]_{opt}$ several times, as discussed in Section 5.3. This failure causes XCS-RC to maintain a higher size of the population. However, assigning an extra procedure in the attempt of eliminating this issue is not considered as a priority. Currently, the parent-picking mechanism prioritizes pairs having lower resemblance. A future improvement might solve this matter, making the system capable of forming a more desirable population than it is currently.



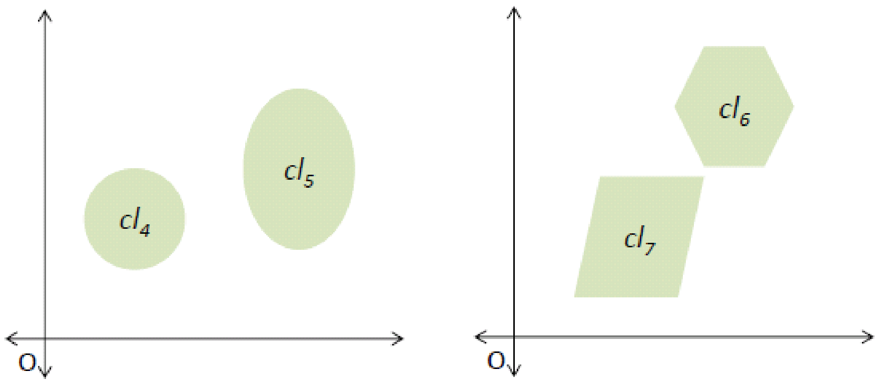
(a) Rectangular

(b) Combining result

Figure 7.4: Classifier coverage in rectangle representations

Another improvement opportunity involves the feature selection capability. A series of experiments shows an indication that the idea is feasible to be implemented. However, the technique might still contain some holes requiring a number of anticipations, e. g., evaluation whether an element is correctly identified as irrelevant. In RC, such functionality plays a similar role to the knowledge correction procedure, anticipating possible mistakes that are made in the past.

After describing some ideas on improving the proposed system, these following topics genuinely reflect some possible steps forward. The first proposed development is inspired by a two-dimensional representation of the cases involving real inputs, e. g., the checkerboard. There, a coverage owned by a pair of rules can be depicted using a Cartesian and the combining result is simply defined as a rectangle between two dots with the longest distance. Figure 7.4(a) shows an instance of the pairing cl_1 and cl_2 while Figure 7.4(b) presents the offspring cl_3.



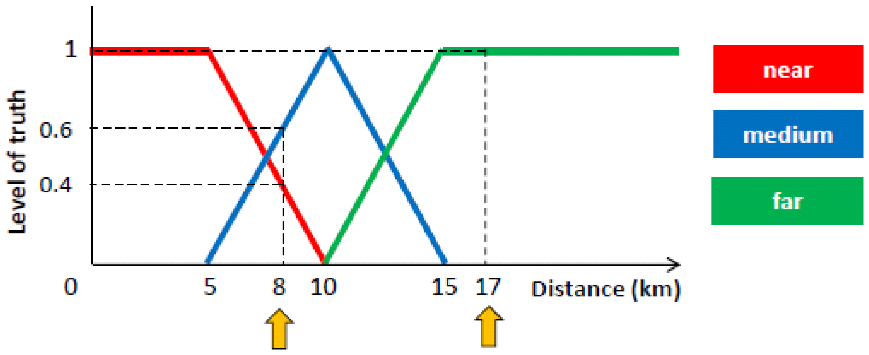
(a) Circular (b) Varying shapes

Figure 7.5: Classifier coverage in other two-dimensional representations

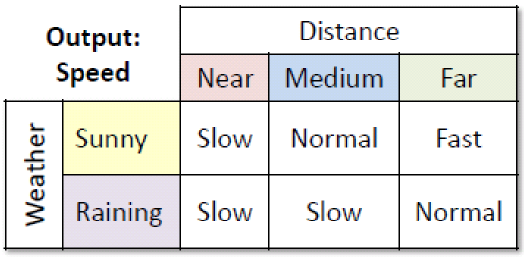
However, such combining cannot easily be implemented to other forms of representation. Figure 7.5(a) illustrates the coverage owned by a pair of classifiers which should be drawn in circular instead of rectangular. Furthermore, other problems might require several forms of two-dimensional representation, instead of them both (see Figure 7.5(b)). It would be interesting to adapt the idea of induction to be implemented to such problems. Several possibilities to approach the combining can be considered, e. g., combinations between mathematical and spatial.

Finally, this thesis ends by presenting a fundamental question: is it possible to improve other learning systems by developing the notions of induction? As a way of thinking that has been developed in several centuries, inductive reasoning may have a huge potential in improving some known techniques. After some learning time, every system possesses its knowledge that can be used to build a conclusion. By having conclusions, the learning process can be sped up and made more efficient. Another possible advantage is increasing the degree of freedom or the autonomy level of other machine learning methods, e. g., in a typical *Fuzzy Logic* learning system as follows.

Fuzzy logic is a system that uses approximations for performing a reasoning operation. The basic mechanism is quite similar to a classifier, i. e., receiving an "IF" (state) and executing a "THEN" part. When an input is received, a so-called *membership function* translates each element into one or more keywords. Figure 7.6(a) provides an example of a membership function that converts real value into one or a combination of three keywords, i. e., "NEAR", "MEDIUM" and "FAR". When the system receives an input (e. g., "8 km"), the membership



(a) Membership function



	Output: Speed	Distance		
		Near	Medium	Far
Weather	Sunny	Slow	Normal	Fast
	Raining	Slow	Slow	Normal

(b) Rule base

Figure 7.6: Typical fuzzy logic components

function translates it into "0.6 MEDIUM and 0.4 NEAR". Another example is given by a state "15 km", which is "100% FAR".

Such translation can be considered a form of classification. Using a series of approximations, Table 7.5 provides a set of one-elemented rules that is able to perform in a similar way to the membership function. Commonly, fuzzy logic copes with states with more than one element, as well as classifier systems. Then, Figure 7.6(b) depicts a reasoning element called *rule base*, which plays a connecting role between the "IF" and "THEN" part of the system.

In the provided instance, the system considers two variables (i. e., distance and weather) to make decisions. Such structure resembles the classifier systems, where the condition covers several input elements by playing the "IF" role and advocates the "THEN" part from the action attribute. E. g., intentionally emphasizing the similarity between Fuzzy Logic and XCS, the rule base operates as *"IF the distance is MEDIUM and the weather is SUNNY, then go NORMAL speed"*.

The attempt to combine fuzzy logic and classifier systems has been investigated and published in several works. In [1], Bonarini introduces the so-called Learning Fuzzy Classifier Systems (LFCS), a system that uses fuzzy sets to interpret the input and output of a Learning Classifier Systems. Then, Cassilas et al.proposes another system called the Fuzzy-XCS, which operates as an XCS with two additional functionalities called "Defector" and "Effector" that fuzzifies and defuzzifies the given values, respectively [10]. The outlook presented here offers a very different approach, i. e., a classifier system that performs the membership function and the rule base component using reinforcement learning cycles.

It is worth to see whether a modified classifier system is able to play a similar role to that owned by fuzzy logic's membership function and rule base. The challenge is provided by an

Table 7.5: Rule representation of the fuzzy logic membership function

No.	Condition	Action	Pred.	No.	Condition	Action	Pred.
1	[0.000; 5.000]	NEAR	1000	13	[11.001;12.000]	MEDIUM	800
2	[5.001; 6.000]	NEAR	800	14	[12.001;13.000]	MEDIUM	600
3	[6.001; 7.000]	NEAR	600	15	[13.001;14.000]	MEDIUM	400
4	[7.001; 8.000]	NEAR	400	16	[14.001;15.000]	MEDIUM	200
5	[8.001; 9.000]	NEAR	200	17	[15.001;50.000]	MEDIUM	0
6	[9.001;50.000]	NEAR	0	18	[0.000; 9.999]	FAR	0
7	[0.000; 4.999]	MEDIUM	0	19	[10.000;10.999]	FAR	200
8	[5.000; 5.999]	MEDIUM	200	20	[11.000;11.999]	FAR	400
9	[6.000; 6.999]	MEDIUM	400	21	[12.000;12.999]	FAR	600
10	[7.000; 7.999]	MEDIUM	600	22	[13.000;13.999]	FAR	800
11	[8.000; 8.999]	MEDIUM	800	23	[14.000;50.000]	FAR	1000
12	[9.000;11.000]	MEDIUM	1000				

obvious difference between both systems. In fuzzy logic, those components are defined solely by human's decision, while XCS tends to autonomously learn using environmental states and responses. Nonetheless, combining the capabilities of two well-known machine learning methods should be an interesting topic, a challenging problem for those who continuously seeks a chance of improvements.

REFERENCES

[1] A. Bonarini. An introduction to learning fuzzy classifier systems. In *Learning Classifier Systems, From Foundations to Applications*, pages 83–106, London, UK, UK, 2000. Springer-Verlag.

[2] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial intelligence*, 40(1):235–282, 1989.

[3] L. Bull. Learning classifier systems: A brief introduction. In L. Bull, editor, *Applications of Learning Classifier Systems*, volume 150 of *Studies in Fuzziness and Soft Computing*, pages 1–12. Springer Berlin Heidelberg, 2004.

[4] M. Butz and S. W. Wilson. An algorithmic description of xcs. In *Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, IWLCS '00, pages 253–272, London, UK, UK, 2001. Springer-Verlag.

[5] M. V. Butz. XCSJava 1.0: An implementation of the XCS classifier system in Java. IlliGAL report 2000027, Illinois Genetic Algorithms Laboratory, 2000.

[6] M. V. Butz, D. E. Goldberg, and K. Tharakunnel. Analysis and improvement of fitness exploitation in xcs: bounding models, tournament selection, and bilateral accuracy. *Evol. Comput.*, 11(3):239–277, 2003.

[7] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in xcs. *IEEE Transactions on Evolutionary Computation*, 8:28 – 46, 2004.

[8] E. Cakar, N. Fredivianus, J. Hähner, J. Branke, C. Müller-Schloer, and H. Schmeck. Aspects of learning in oc systems. In C. Müller-Schloer, H. Schmeck, and T. Ungerer, editors, *Organic Computing — A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, pages 237–251. Springer Basel, 2011.

[9] E. Cakar and C. Müller-Schloer. Self-organising interaction patterns of homogeneous and heterogeneous multi-agent populations. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on*, pages 165–174, 2009.

[10] J. Casillas, B. Carse, and L. Bull. Fuzzy-xcs: A michigan genetic fuzzy system. *Fuzzy Systems, IEEE Transactions on*, 15(4):536 –550, aug. 2007.

[11] N. Fredivianus, K. Kara, and H. Schmeck. Stay real!: Xcs with rule combining for real values. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, pages 1493–1494, New York, NY, USA, 2012. ACM.

[12] N. Fredivianus, H. Prothmann, and H. Schmeck. Xcs revisited: A novel discovery component for the extended classifier system. In K. Deb, A. Bhattacharya, N. Chakraborti, P. Chakroborty, S. Das, J. Dutta, S. Gupta, A. Jain, V. Aggarwal, J. Branke, S. Louis, and K. Tan, editors, *Simulated Evolution and Learning*, volume 6457 of *Lecture Notes in Computer Science*, pages 289–298. Springer Berlin / Heidelberg, 2010.

[13] N. Fredivianus, U. Richter, and H. Schmeck. Collaborating and learning predators on a pursuit scenario. In M. Hinchey, B. Kleinjohann, L. Kleinjohann, P. Lindsay, F. Rammig, J. Timmis, and M. Wolf, editors, *Distributed, Parallel and Biologically Inspired Systems*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 290–301. Springer Boston, 2010.

[14] A. W. Hirshfeld. *Parallax: The Race to Measure the Cosmos*, volume 33 of *Bulletin of the American Astronomical Society*. Henry Holt and Company, May 2001.

[15] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern directed inference systems*, pages 313–329. Academic Press, 1978.

[16] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International*, pages 121–129. Morgan Kaufmann, 1994.

[17] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[18] T. Kovacs. Deletion schemes for classifier systems. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 329–336. Morgan Kaufmann, 1999.

[19] T. Kuhn. *The Copernican Revolution: Planetary Astronomy in the Development of Western Thought*. A Harvard Paperback. Harvard University Press, 1957.

[20] P. L. Lanzi. An analysis of generalization in the xcs classifier system. *Evol. Comput.*, 7(2):125–149, 1999.

[21] B. Lenzitti, D. Tegolo, and C. Valenti. Prey-predator strategies in a multiagent system. In *Computer Architecture for Machine Perception, 2005. CAMP 2005. Proceedings. Seventh International Workshop on*, pages 184–189, July 2005.

[22] G. Lloyd. *Early Greek science: Thales to Aristotle*. Ancient culture and society. Chatto & Windus, 1970.

[23] D. O'Leary. *How Greek Science Passed to the Arabs*. Ares Publishers, 1949.

[24] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In C. Hochberger and R. Liskowsky, editors, *INFORMATIK 2006 ? Informatik für Menschen!*, volume P-93 of *LNI*, pages 112–119. Bonner Köllen Verlag, Oktober 2006.

[25] U. Richter, H. Prothmann, and H. Schmeck. Improving xcs performance by distribution. In X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. Tan, J. Branke, and Y. Shi, editors, *Simulated Evolution and Learning*, volume 5361 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin Heidelberg, 2008.

[26] G. Saliba. *Islamic Science and the Making of the European Renaissance.* History of science: Islamic studies. MIT Press, 2007.

[27] H. Schmeck. Organic computing - a new vision for distributed embedded systems. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 201–203, Washington, DC, USA, 2005. IEEE Computer Society.

[28] H. Schmeck, C. Müller-Schloer, E. Cakar, M. Mnif, and U. Richter. Adaptivity and self-organization in organic computing systems. *ACM Transactions on Autonomous and Adaptive Systems*, 5(3):10:1–10:32, 2010.

[29] B. Steffens. *Ibn Al-Haytham: First Scientist.* Profiles in Science. Morgan Reynolds Incorporated, 2007.

[30] C. Stone and L. Bull. For real! xcs with continuous-valued inputs. *Evol. Comput.*, 11:299–336, September 2003.

[31] R. Taton, C. Wilson, and M. Hoskin. *The General History of Astronomy: Volume 2, Planetary Astronomy from the Renaissance to the Rise of Astrophysics.* General History of Astronomy. Cambridge University Press, 1995.

[32] A. M. Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[33] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, England, 1989.

[34] M. Weidhorn. *The Person of the Millennium: The Unique Impact of Galileo on World History.* iUniverse, 2005.

[35] S. W. Wilson. Classifier fitness based on accuracy. *Evol. Comput.*, 3(2):149–175, 1995.

[36] S. W. Wilson. Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[37] S. W. Wilson. Get real! xcs with continuous-valued inputs. In *Learning Classifier Systems, From Foundations to Applications*, pages 209–222, London, UK, 2000. Springer-Verlag.

APPENDIX

Files containing the results of the experiments along with the source code for XCS and XCS-RC are available at http://www.nuggstory.com/diss/ for downloads.