Titre: Title:	Comparison of Adaptive Behaviors of an Animat in Different Markovian 2-D Environments Using XCS Classifier Systems
Auteur: Author:	Armin Najarpour Foroushani
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Najarpour Foroushani, A. (2013). Comparison of Adaptive Behaviors of an Animat in Different Markovian 2-D Environments Using XCS Classifier Systems [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1194/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1194/
Directeurs de recherche: Advisors:	Jean-Jules Brault
Programme: Program:	génie électrique

UNIVERSITÉ DE MONTRÉAL

COMPARISON OF ADAPTIVE BEHAVIORS OF AN ANIMAT IN DIFFERENT MARKOVIAN 2-D ENVIRONMENTS USING XCS CLASSIFIER SYSTEMS

ARMIN NAJARPOUR-FOROUSHANI

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

Juin 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

COMPARISON OF ADAPTIVE BEHAVIOR OF AN ANIMAT IN DIFFERENT MARKOVIAN 2-D ENVIRONMENTS USING XCS CLASSIFIER SYSTEMS

présenté par : NAJARPOUR-FOROUSHANI Armin

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. LE NY Jérôme, Ph.D., président

M. BRAULT Jean-Jules, Ph.D., membre et directeur de recherche

M. PARTOVI NIA Vahid, Ph.D., membre

DEDICATION

To my parents who taught me hard work

To my grandmother for her encouragement

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor of research, Mr Jean-Jules Brault who provided me advices and supervision to successfully complete this project. Thanks to Professor Jerome Le Ny and Professor Vahid Partovi Nia for their helpful guidance and comments on my thesis.

Special thanks to my parents who gave me unconditional love, patience, and encouragement during this work.

RÉSUMÉ

Le mot "Animat" fut introduit par Stewart W. Wilson en 1985 et a rapidement gagné en popularité dans la lignée des conférences SAB (Simulation of Adaptive Behavior: From Animals to Animats) qui se sont tenues entre 1991 à 2010. Comme la signification du terme "animat" a passablement évoluée au cours de ces années, il est important de préciser que nous avons choisi d'étudier l'animat tel que proposée originellement par Wilson.

La recherche sur les animats est un sous-domaine du calcul évolutif, de l'apprentissage machine, du comportement adaptatif et de la vie artificielle. Le but ultime des recherches sur les animats est de construire des animaux artificiels avec des capacités sensorimotrices limitées, mais capables d'adopter un comportement adaptatif pour survivre dans un environnement imprévisible. Différents scénarios d'interaction entre un animat et un environnement donné ont été étudiés et rapportés dans la littérature. Un de ces scénario est de considérer un problème d'animat comme un problème d'apprentissage par renforcement (tel que les processus de décision markovien) et de le résoudre par l'apprentissage de systèmes de classeurs (LCS, Learning Classification Systems) possédant une certaine capacité de généralisation. L'apprentissage d'un système de classification LCS est équivalent à un système qui peut apprendre des chaînes simples de règles en interagissant avec l'environnement et en reçevant diverses récompenses.

Le XCS (eXtended Classification System) introduit par Wilson en 1995 est le LCS le plus populaire actuellement. Il utilise le Q-Learning pour résoudre les problèmes d'affectation de crédit (récompense), et il sépare les variables d'adaptation de l'algorithme génétique de celles reliées au mécanisme d'attribution des récompenses.

Dans notre recherche, nous avons étudié les performances de XCS, et plusieurs de ses variantes, pour gérer un animat explorant différents types d'environnements 2D à la recherche de nourriture. Les environnements 2D traditionnellement nommés WOODS1, WOODS2 et MAZE5 ont été étudiés, de même que des environnements S2DM (Square 2D Maze) que nous avons conçus pour notre étude. Les variantes de XCS sont XCSS (avec l'opérateur "Specify" qui permet de diminuer la portée de certains classificateurs), et XCSG (avec la descente du gradient en fonction des valeurs de prédiction). Nous avons constaté une amélioration sensible de leur performance d'apprentissage.

Nous avons proposé une version combinant XCSS et XCSG, appelée XCSSG. La comparaison des résultats montre que pour des environnements simples tels que WOODS1 et WOODS2, les performances de tous les algorithmes (soit le nombre d'étapes que l'animat doit faire pour atteindre la nourriture) déjà proposés sont très proches, mais que dans des environnements plus complexes tels que MAZE5, l'approche XCSSG converge rapidement près de la solution optimale (nombre minimum d'étapes).

Pour étudier la capacité d'apprentissage de XCS et ses variantes sur une plus grande variété d'environnements (markoviens et non markoviens) que les environnements classiques WOODSx et MAZEy, nous avons conçu un générateur d'environnements S2DM. Les différents algorithmes XCS étudiés ont été testés sur ces environnements et les résultats montrent clairement que les capacités d'apprentissage des différents XCS s'approchent toutes des performances optimales. De plus, une analyse de l'évolution du nombre de classificateurs/règles d'une population a également été faite pour mieux illustrer les capacités de généralisation de chacun des algorithmes XCS.

Nous avons finalement proposés trois nouveaux scénario pour étudier les variations de populations de classificateurs des différents XCS. D'abord, un scénario où les ressources se déplacent légèrement. Puis, un scénario compétitif inter-espèces (XCS vs XCSSG) pour le partage d'une ressource commune. Ce scénario est basé sur les équations de Lotka-Volterra et permet de comparer dynamiquement les performances des deux algorithmes. Un troisième scénario a été proposé faisant intervenir un animat ayant des capacités supérieures de vision afin d'étudier la possibilité d'apprendre dans des environnements non-markoviens pour un animat classique, mais markoviens pour un animat moins myope. Les résultats de ce troisième scénario ne sont pas ceux auxquels nous nous attendions. En effet, l'animat n'a pas su profiter de cette supériorité pour améliorer ses performances. C'est pour nous un problème ouvert que nous nous proposons d'explorer dans une nouvelle recherche.

ABSTRACT

The word “Animat” was introduced by Stewart W. Wilson in 1985 and became popular since the SAB line conferences “Simulation of Adaptive Behavior: from Animals to Animats” that were held between 1991 and 2010. Since the use of this word in the scientific literature has fairly evolved over the years, it is important to specify in this thesis that we have chosen to adopt the definition that was originally proposed by Wilson.

The research on animat is a subfield of evolutionary computation, machine learning, adaptive behavior and artificial life. The ultimate goal of animat research is to build artificial animals with limited sensory-motor capabilities but able to behave in an adaptive way to survive in an unknown environment. Different scenarios of interaction between a given animat and a given environment have been studied and reported in the literature. One of the scenarios is to consider animat problems as a reinforcement learning problem (such as a Markov decision processes) and solve it by Learning Classifier Systems (LCS) with certain generalization ability. A Learning classifier system is equivalent to a learning system that can learn simple strings of rules by interacting with the environment and receiving diverse payoffs (rewards).

The XCS (eXtented Classification System) [1], introduced by Wilson in 1995, is the most popular Learning Classifier System at the moment. It uses Q-learning to deal with the problem of credit assignment and it separates the fitness variable for genetic algorithm from those linked to credit assignment mechanisms.

In our research, we have studied XCS performances and many of its variants, to manage an animat exploring different types of 2D environments in search of food. 2D environments traditionally named WOODS1, WOODS 2 and MAZE5 have been studied, as well as several designed S2DM (SQUARE 2D MAZE) environments which we have conceived for our study. The variants of XCS are XCSS (with the *Specify* operator which allows removing detrimental rules), and XCSG (using gradient descent according to the prediction value).

We have proposed a version combining XCSS and XCSG called XCSSG. The comparison of results shows that for simples environments such as WOODS1 and WOODS2, the performance (the number of steps that the animat must follow to reach the food) of all previously proposed

algorithms are very close, but in more complex environments such as MAZE5, the proposed approach of XCSSG converges rapidly to near optimal solution (minimum number of steps).

To investigate the learning ability of XCS and its variants on a higher variety of environments (Markovian or non-Markovian) than the classic environments WOODSx and MAZEy, we have conceived an environment generator S2DM. Different XCS-family algorithms studied have been tested on these environments and the results clearly show the ability of XCS-family in learning all of these new environments and approaching the optimal performances. Furthermore, an analysis of the evolution in the number of classifiers/rules in a population set has also been done to illustrate the ability of XCS-family algorithms in producing general rules (generalization).

We have finally presented three new scenarios to study the variations of population sets of different XCS classifiers. First of all, a scenario where resources shift gently. Then, an inter-species competitive scenario (XCS and XCSSG) for sharing of a common resource. This scenario is based on Lotka- Volterra equations and allows to dynamically compare the performances of the two algorithms. A third scenario has been proposed involving an animat with higher vision abilities to investigate the ability to learn in non- Markovian environments for a classic animat but that become Markovians when the animat can perceive on a farther distance. The results of this third scenario are not the ones that we were expecting. Indeed, the animat does not take advantage of its visual superiority to improve its performance. For us it is an open problem that we intend to explore in a new search.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABREVIATION

AB	Adaptive Behavior
ACS	Anticipatory Classifier Systems
AI	Artificial Intelligence
Alife	Artificial Life
ARP	Action Reply Process
CAS	Complex Adaptive Systems
CXCS	Corporate XCS
GA	Genetic Algorithm
GP	Genetic Programming
LCS	Learning Classifier Systems
MDP	Markov Decision Process
NSF	Number of Steps to Food
RL	Reinforcement Learning
SB-XCS	Strength-based XCS
XCS	EXtended Classifier Systems
XCSF	XCS for function approximation
XCSG	Gradient-Based XCS
XCSI	XCS for integer inputs
XCS-LP	XCS with continuous reinforcement
XCSM	XCS with addition of Memory
XCSS	XCS with *Specify*
XCSSG	Gradient-based XCS with *Specify*
X-NCS	Neural XCS

ZCS	Zeroth-level classifier systems
S2DM	Square 2 Dimensional Maze

LISTE OF APPENDICES

INTRODUCTION

Motivation

The concept of “Animat” was invented by Stewart W. Wilson in 1985 by publishing the paper “KNOWLEDGE GROWTH IN AN ARTIFICIAL ANIMAL [2].” Using this word became popular after conference “Simulation of adaptive behavior: from animals to animats (SAB90)” in 1990 in Paris. After three conferences, the International Society for Adaptive Behaviour was formed that contains many contributions related to the animat approach. They have a journal, Adaptive Behaviour and a proceeding which is published every two years.

In debates about artificial intelligence, several researchers believed that recreating the human intelligence as a purpose is a very far and doubtful goal, and it would be better to first understand basics and simpler capacities of intelligence that are common between human and animals while interacting with the environment, such as their adaptive behavior for foraging, navigation and obstacle avoidance. According to these debates two important things were considered: inspiration from biology and applying the bottom-up approach to *AI* (Artificial Intelligence). Wilson suggested using of animal models of increasing complexity and synthesize them to study natural and artificial intelligence [2]. Using the animal models to study intelligence depending on the complexity of the model or complexity of the animal can lead to intelligence at its primitive levels or more complex levels such as human. The primitive animal models give a good insight into the basis of intelligence in general. They solve basic problems which are common among a wide range of animals from the simplest ones such as C. elegans to the most complex ones such as human being. The behavioral models of simple animals are based on solving these problems. These behavioral models help us to understand the whole intelligence and design more complex models.

Based on [2] the simple animals have four common basic characteristics:

1. Animals at each moment receive only some sensory signals from the environment which are important at that moment.

2. Animals have the ability of performing action to change these environmental signals.

3. Existence or absence of certain signals such as food consumption has special meaning for animals.

4. Animals act to optimize the rate of occurrence of certain signals. This action is produced by an internal and external operation.

1 and 2 are related to sensory-motor system of animals and 3 and 4 are related to the notion of "need". Wilson called the artificial animals that follow these four rules "animat".

Animat Approach

The animat approach is sub-category of evolutionary computation, machine learning, adaptive behavior, and artificial life. *Artificial life* or *Alife* investigates the logic and formal basis of life and living systems to understand the complex information processing in these systems and tries to simulate or synthesize based on these bases. Emergent property is central to alife research. It is a property that a system and its properties (a "whole") as the interaction of its parts has a global behavior that can't be understood of its parts [3]. Actually, alife focuses on those complex systems that are inspired from life [4]. Alife is a bottom-up (synthetic) approach constructing life from its basic elements. Adaptive behavior is the behaviour in a changing and unknown environment for survival that can change in response to agent's environment [5].

Animats are artificial animals. They can be simulated animals or physical robots. The definition of the animat approach is:

Understanding the formal basis of animals' life and synthesize it in a form of an artificial animal in a changing and uncertain environment to provide understanding of adaptive behavior of animals for surviving in artificial and real world.

Life of animat is considered as its adaptive behaviour which is the interaction between animat and the environment for surviving, thus, environmental complexity has effect on the adaptive behavior of the animat. Complex adaptive behaviors are the result of complex environments. So, a general model of interaction between agent (animat) and environment needs a general theory of environment. Wilson in [6] introduced a general theory of environment based on finite state

machines. The general theory of environment can be a dynamic system model too, i.e. the behavior of agent in an environment is a dynamic system, where a state is the condition of animat at a given time and its dynamic determines the state change [7]. Two capabilities are central to animat approach: sensing the environment and action. These abilities together are considered a sensory-motor system. Animats search for essential sensory information and select actions to perform beneficially in the environment [3]. Sensory system links the agent to environment and actions allow it to behave adaptively [5]. Adaptive behavior is the consequence of actions that animat performs based on the sensory information from the environment and application of a control algorithm (control architecture). Needs are the main drivers of animal behavior and can be regarded the root of intelligence. The concept of needs is common from human to very simple animals, i.e. all of them have a number of needs. To satisfy needs animat has to live in the environment and the complexity of environment influences the complexity of its behavior and the performance of its operation.

The long-term goal of animat approach is to understand human intelligence incrementally, i.e. starting from simple environments and increasing the complexity of environments and architectures by adding necessary features (bottom-up approach). The meaning of "incrementally" is increasing the complexity of needs or complexity of environment to determine change in the animat behavior necessary to satisfy the needs [6].

Animat Approach and AI

AI is the synthetic and computational study of intelligence. AI includes two approaches to deal with the problems of agent behaving in the environment: standard AI and Behavior-based AI. Standard AI concerns with the competition of machine with human by simulation of the abilities of human cognition in the form of computer programs that are connection of symbols in internal reasoning that yield external stimuli [6]. Standard AI was popular until near 1990. In behavior-based AI agent interacts with the environment through sensing and making action.

The behavior-based AI emerged against the limitations of the standard AI in which uses symbol-based tasks and ignores sensory information, needs, perception, adaptation, learning, and coping with the environment. The standard AI is limited for controlling of a physical agent in an environment and has a big processing delay when interacts with an unknown environment, and therefore, it is limited for understanding of intelligence.

The animat approach is a behavior-based approach which considers interaction with the environment through sensing and action. Its aim is to simulate and understand complete animal-like systems at simple level and reach to human intelligence "from below" incrementally.

Reinforcement learning description of the animat problem

Animat problem can be described in several ways. One way is the problem of an animat in the environment containing payoff (reward or punishment) that are given to each action that animat performs. In this kind of problem the animat tries to learn and maximize its total reward by searching the environment. Among several methods to solve a reinforcement learning problem, learning classifier systems have the ability of generalization (ability of the system to reach to a rule for assigning of each action to each state more general than having a table for assignment of actions to all states). Learning classifier systems learn the payoff environment by a set of rules called classifiers. Among different learning classifier system methods, XCS that was introduced by Wilson (1995) is the most popular and has better performance and generalization ability in comparison to the other learning classifier systems methods. Animat problems can be represented in a framework to be solved with XCS classifier systems. The developed models of XCS for more complex Markovian environments are XCS-with-*Specify* (XCSS) and gradient-based XCS (XCSG). XCSS removes rules with mal-functionalities and XCSG presents a gradient-based prediction of reward to improve the performance of XCS.

Objectives

The objective of this thesis is to solve a reinforcement learning-based animat problem using XCS classifier systems and compare the performance in different 2-D environments. The contribution of this work is the presentation of a new method that is a combination of XCS-with-*Specify* operator (XCSS) and gradient-based XCS (XCSG) that is called XCSSG to improve the performance and speed of the system. A comparison between performance of several developed models of XCS such as XCS, XCSS, XCSG, residual XCSG, and XCSSG is done in this thesis. Study of the effect of the subsumption mechanism (a mechanism that removes useless rules of the system) on the performance of XCS in various Wilson's animat problems in different environments is also presented. Other contributions of this work are introducing new maze environments beyond the traditional environments that are presented in the literature and trying to solve them using XCS-family algorithms (XCS, XCSG, XCSS, XCSSG, XCS with

subsumption). Introducing an unstable resource problem with XCS animat to test the ability of XCS to adapt to a changing environment is presented in this work. A competitive platform for comparison of XCS and XCSSG is introduced based on Lotka-Volterra equation to introduce new way for comparison of two adaptive algorithms. An animat with higher vision abilities is also introduced in this thesis to provide conditions to convert a non-Markovian environment to a Markovian environment for the animat and let XCS and XCSSG to learn with these new sensory abilities. In Chapter 1 first the animat problem and its basic components are described and Wilson's animat that is a particular kind of reinforcement learning (RL) animat will be introduced. It is followed in Chapter 2 by providing an introduction to the mathematical description of a reinforcement learning problem, methods to solve it, and description of learning classifier systems. In Chapter 3 XCS is introduced as the main method in this thesis to deal with the Wilson's animat problem and it finishes by a literature review on XCS animat. To use XCS in more complex environments and improve its performance, XCSS, XCSG, and their combination (XCSSG) are introduced in Chapter 4. At the end of this chapter a comparison of different methods and also their comparison with Q-learning are made to compare the work with the older basic methods. To study the abilities of XCS beyond the traditional works on XCS, in Chapter 5 new environments are presented and new scenarios are introduced to test the ability of the XCS animat in operating in new situations. Results of learning XCS are then compared and conclusions are presented.

CHAPITRE 1 ANIMAT PROBLEM

In this chapter the basic components of animat problem and the role of each component are introduced. The concept of Reinforcement learning animat and Wilson's animat are introduced and used as the basis of animat problem in this thesis.

1.1 Structure of the animat problem

Animat problem is a problem that is expressed based the formal basis of animals' life in which an agent interacts adaptively with an unknown environment to survive. Formal basis of animals' life differs for different animals. However, there are basic rules that are common between all of them, from the simplest one to human intelligence and are considered as the basic rules of intelligence in animals. These basic rules are categorized into two groups: 1) having sensory-motor system and 2) having needs. These two properties construct the common basis of animat problem. Sensory and motor systems are connected by a control architecture that in its simplest form is a reflex, but can perform a more complex functionality such as learning or evolution. Control architecture connects sensing and action by a mapping for the purpose of surviving (e.g. food seeking). Interaction of animat and environment for survival has its root in satisfaction of needs. Depending on the needs that have been considered in an animat problem, environment can be different and the corresponding surviving task to satisfy these needs is different. For example finding food, avoiding obstacles, and wall tracking are various kinds of surviving tasks that are different for different environments. Animat interacts with the environment through sensing and action to satisfy its needs. Adaptive behavior is the result of interaction between animat and environment. An abstract diagram representing the basic architecture of animat problem is shown in Figure 1-1.

Long term goal of the animat approach is bottom-up understanding of intelligence that is starting from primary levels of intelligence (simple animals with minimal architecture in simple environments) and increasing complexity of problem until reaching to human intelligence. So, more components can be added to the basic architecture of the animat problem to make it appropriate for more complex environments.

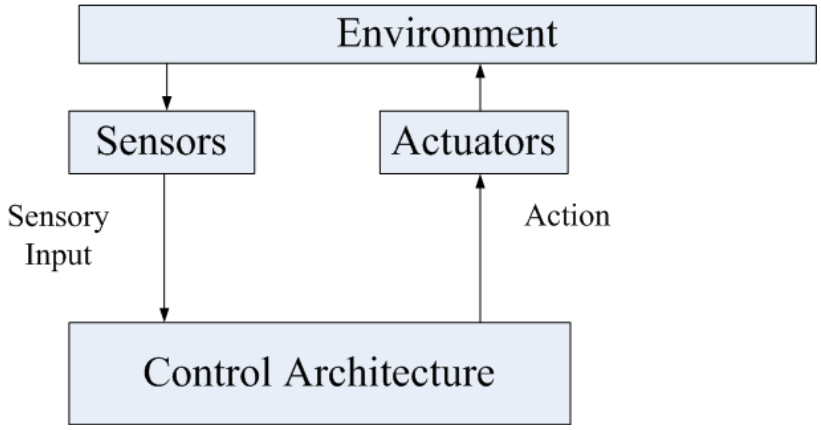


Figure 1-1: Basic block diagram of an animat problem. Animat interacts with the environment to satisfy its needs.

1.1.1 Components of an animat problem

Based on the definition of the animat problem the basic components of an animat problem are as follows:

- Formal basis of animals' life
- Environment
- Adaptive behavior

Formal basis of animals' life are the bio-inspired rules based on real rules of the life of animals and describing the life of an animat and its interaction with the environment. Formal basis of animals' life are usually general rules that are common between all types of animals from the simplest one such as fruit fly to the most complex one such as human. These bases are classified into two main groups that are common among every kind of animals: i) having sensory-motor system and ii) having needs. Sensory-motor system consists of sensors to sense the environment and actuators to do action and change sensory signals. Control architecture maps sensory information to the action. This mapping can be a simple reflex or a more complex mapping such as learning. The animat interacts with the environment to satisfy its needs (present or future). This interaction is via the sensory-motor system and the objects that satisfy its needs and are available in the environment. The Animat can be a physical robot or simulated animal in the environment. The body, number and position and type of sensors, number and position and type

of actuators, and the way of connection of these components to the control architecture are significant for the adaptive process. In addition the constraints that are regarded on the animat's body such as the type of legs or the shape of body can affect the adaptive behavior. So, the word of "embodied" is applied when role of the body is considered important for the adaptive behavior.

Environment is a physical or simulated world containing food or other objects necessary for the need satisfaction (survival). The environment mainly is the simulation of animals' ecosystem and is created by inspiration from real ecosystem. Based on the animat's needs that are regarded for a specific problem an environment is designed and the surviving tasks are assigned. Examples of surviving tasks in various animat problems are acquiring maximum resources of food, reaching to a particular cell, reaching to the first food, maintaining minimum level of energy, living as long as possible, foraging (food seeking), prey hunting, and obstacle avoidance.

Complexity of the environment can be characterized by setting of tasks and its pattern of objects. For example distribution of food (in foraging task) and obstacles determines the complexity of surviving task for some kind of animat problems. So, a formal theory of environment can be used to give a better insight into the complexity of environment. A formal theory of environment can be expressed by a finite state machine (FSM) model [6]. In this model actions are input to the environment and sensory stimuli are output. For a given input the number of possible outputs is finite. The model is expressed by:

$$Q(t+1) = F(Q(t), A(t))$$

$$E(t+1) = G\big(Q(t), A(t)\big) \qquad (1.1)$$

Where A is the action, E is the sensory stimulus, Q is the current state of the perceived environment, and t is discrete time. F is a function that represents the change of state of the environment to the next state (transition function) for action at time-step t and G is a function that represents the sensory stimulus at state $Q(t)$ for action at time-step t. The model says that the action in an environment results a new sensory stimuli. It also can be concluded that the same action inputs to different situations of the environment result in different sensory stimuli. This model is also used to provide a measure for the level of complexity [6]. If the animat is equipped

with more sensors in a certain environment, it can see more details of the environment and may adapt easier.

Two classes of environments based on the state transition of an agent (that is situated in the environment) are definable: Markovian environments and non-Markovian environments. Markovian environments are those environments that the best action in a state can be determined by having the sensory information in current state. Non-Markovian environments are environments that the best action in a state is not determinable only from the sensation vector in current state. In other words, for non-Markovian decision process information from the states that it has passed before, or may be all of them are needed.

Adaptive behavior is the result of internal cognitive process of animat and its interaction with the environment [8]. It is a behavior for need satisfaction (surviving) in an unknown environment. The surviving of animat depends on the ability of animat to cope with the environment through experience. This ability is different depends on the complexity of environment, the surviving task that is based on the regarded needs, the control architecture, number, position, and type of sensors and actuators. Control architecture has a central role in the adaptive behavior. It maps sensory information to the action and the sequence of actions constructs the adaptive behavior of animat. Based on [3] and [9], [10], and [11] different kinds of control architectures (adaptive behaviors) are as follows:

1. Programmed behavior :

Programmed behavior is the result of a control architecture that is designed for a certain purpose. For example, in a population of animats all of them can have the same architecture and the architecture has been constructed from several layers each composed of networks of finite state machine. This kind of architecture is designed to decompose complicated architectures into simple modules each perform a simple behavior. The modules are organized in different layers that each layer implements a certain goal of agent. Higher layers are more abstract and work to reach to the overall goal. This approach is a bottom-up approach. The programmed behavior can be used for blind robots that operate without sensory information from the environment.

2. Learning:

Learning is the process of building a general model based on a set of seen examples and using that model for prediction in new unseen situations. Importance of learning is in its application to noisy, changing, and unknown environments where animat has to decide what to do in new situations in the environment. In learning animat obtains knowledge by direct interaction to the environment via sensors [12]. Based on the literature three important learning techniques for animat are as follows:

- *Unsupervised Learning*: is a kind of learning that agent (or animat) learns and reconstruct patterns by associating different parts of the pattern with the other parts. For example using Kohonen neural network, a robot would be able to recognize different structures of the environments by finding the similarities that it uses to cluster. So, in this way the robot can move in the environment and categorize it.
- *Reinforcement Learning*: learning to behave by receiving payoff from the environment and trying to maximize the total amount of expected payoff. An environment for animat problem can be accounted as a reinforcement learning problem which animat tries to learn. For example, Markovian environments are formulated as a Markov Decision Process (see 2.1.2.3) that is in fact a reinforcement learning problem. To solve a reinforcement learning problem several techniques such as dynamic programming, temporal difference, Q-learning, bucket brigade algorithm, and as we will see learning classifier systems can be applied.
- *Associative Learning*: in associative learning animat makes a cognitive map of the environment. Cognitive map is a map that animat memorizes. This map associates the sensory information to actions for the navigation task. For animals the cognitive maps contain topological and metric information about the environment that they have learned to determine. The spatial representation of the environment is encoded in their hippocampus which is part of the animals' brain to help them survive in the environment.
- *Conditioning*: a number of learning processes that improve perception or motor skills in animals by perception without need for higher cognitive processes.

3. Evolution:

Evolution is the process of improving behavior of individuals in a population. The improvement performs by selecting the individuals that have been adapted and removing individuals that have not been adapted well. With a simple evolutionary rule it can generate an unpredictable or very complex behaviour that is not planned [13]. The evolution often is based on natural selection models. For animat problem evolutionary strategies that are usually applied are genetic algorithm, genetic programming, evolving control parameters of neural networks with GA or GP, evolution of control program, evolutionary programming, and evolution strategies.

4. Development:

In artificial evolution the genotype of an individual is decoded and transformed into a phenotype. In nature, interaction of genetic information and environment builds the phenotype of an animal. This process is called development and here a bio-inspired developmental architecture can be considered for animat. In development architectures connections between sensory and motors neurons is possible. The structure and function of these neurons are designed by human. Geometrical nature of the developmental system and the animat's body is important to build and connect neural modules. The development architecture has been used to evolve a neural network to control the locomotion of a 6-legged animat[14].

5. Combination of different forms of control architectures is possible. Examples are as follows:
    - Evolution based learning techniques
    - Evolution of neural controller
    - Neural controllers that are built incrementally at run time using RL techniques
    - Recurrent neural networks learning using back-propagation
    - Self-organizing neural networks.

1.2 Choice of the animat problem

Research in the animat context can be performed on problem as a whole with consideration of all details or can be focused more on one specific component. Subject of different researches in animat context based on [15] are: Adaptive behavior, Perception and motor control, Architecture,

Action selection and behavioral sequences, Internal world model for navigation, Learning, Evolution, External environment, Collective and social behaviors, and Applied adaptive behavior. Depending on the considered details in each subject a variety of tasks and problems are available. So, it is clear that the animat problem can be represented in different ways.

One of way for representation of animat problem is reinforcement learning approach. *Reinforcement learning (RL)* is a form of machine learning, in which an agent operates in the environment by receiving reward. The final goal of agent is maximization of the total rewards. The animat problem in this way can be expressed in the RL framework: action, sensing the environment, state, and reward (such as obtaining a food or reaching to an obstacle).

In RL context, environment can be Markovian, non-Markovian, or any combination of them. The definition of environment in reinforcement learning depends on the important features that are considered in a certain problem. In the case of Markovian environments RL problem is expressed as a Markov decision process. For Markovian environments Q-learning (see 2.1.3.1), learning classifier systems, and dynamic programming methods can be used in different ways for an animat to survive. For non-Markovian problems there is no exact method to solve. We call the animat problem that is represented in the reinforcement learning framework "RL animat". The block diagram of a typical RL animat is shown in Figure 1-2.

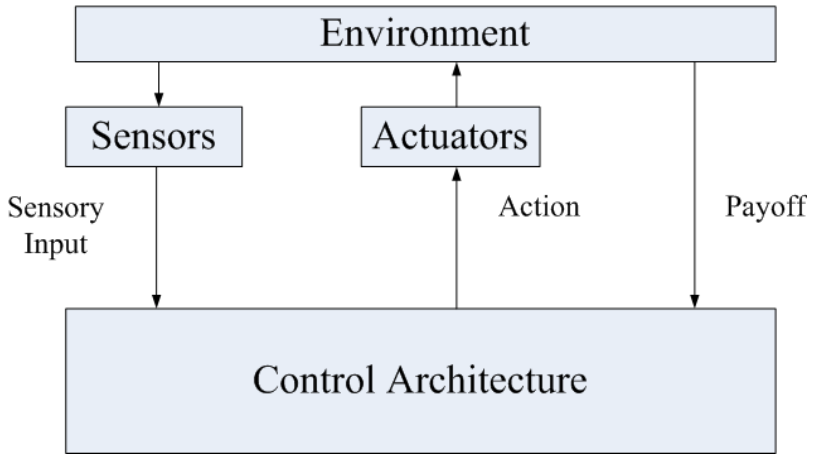


Figure 1-2: Block diagram of RL animat problem learns by means of payoff from environment.

Learning classifier systems (LCS) have generalization capability and are applicable for large and complex problems where Q-learning alone cannot be used because it needs a high amount of

memory and doesn't have generalization ability. For this reason, in this project LCS is applied to deal with the animat problem. We call "LCS animat" or "Wilson's animat" to refer to a RL animat problem that LCS is used as its control architecture.

1.3 Wilson's animat

Wilson studied learning of animat in the environment using learning classifier systems that is a specific type of RL animat problem [2]. The block diagram of Wilson's animat is illustrated in Figure 1-3. It is specific type of RL animat that the control architecture is a learning classifier systems algorithm. The environment that he considered for the animat was a rectangle with 18 rows and 58 columns that was continued toroidally at the edges and was called woods7 [2] (see Appendix-1). In woods7 at various positions there exist objects which are represented by T and F and b in which Ts are obstacles, Fs are foods, and bs are empty places. At each position animat senses 8 cells around it and stores them in a sense vector which is clockwise representation of these positions starting from the top. This vector is composed of Fs, Ts, and bs. For each of these objects an internal two bits representation is considered, 11 for F, 01 for T, and 00 for b. So, a 16 bit sense vector represents the animat's sensory information at each time step. This 16 bits sense vector is called the detector vector. For example $TTbbFbbb = 0101000011000000$. Detector vector will be used as the input for the process of LCS control architecture in animat. A number between 0-7 which represents one step movement to one of the 8 available directions is considered as an action. The action numbers are constructed clockwise starting from the top (see Figure 1-4). The movement is toward a position which may contain an object. If the movement is toward 00, the animat will receive no signal. If the movement is toward 01, the step won't be allowed because it's an obstacle. If the movement is toward 11, the animat will receive a reward signal. The goal of Wilson's animat is learning to find a food, i.e. after finding a food the process starts again from a random blank point in the environment and after a lot of iterations from different starting points, the number of steps to food reduces to a stable value. Wilson made a reinforcement learning model of animat problem and solved it using learning classifier systems (LCS). The LCS mechanism uses the reward from the environment. So, at each step that the animat eats a food; a reward is given to him that is used in the LCS mechanism.

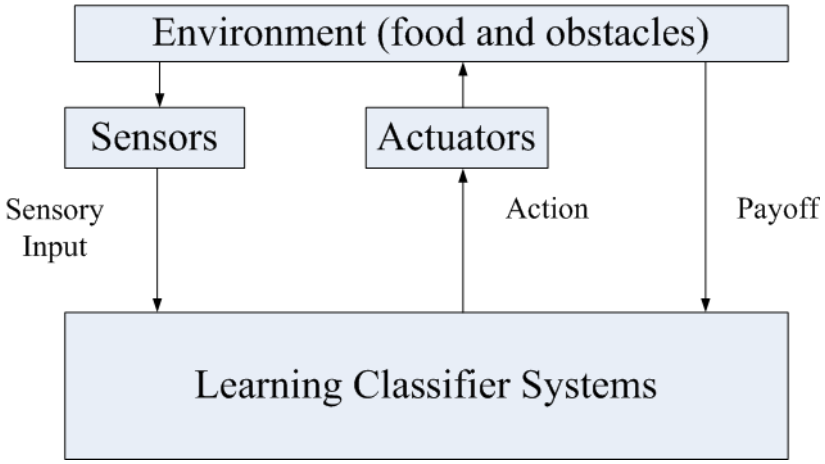


Figure 1-3: Block diagram of Wilson's animat learns by means of payoff from environment

7 0 1

6 * 2

5 4 3

Figure 1-4: Directions defined for the sensation and the movement of the Wilson's animat. * is the animat and 0-7 shows the consequence of the sensory vector and also the codes of directions that the animat can move.

In learning classifier systems the association between sensing and action is represented by condition-action rules. The condition matches the aspects of local environment and the internal state and action determine the internal state. This association are learned by the animat. The basic problem of LCS animat is the generation of the rules to take an appropriate action to optimize the rate of occurrence of certain signals. So, the first step is rule discovery, second step is keeping the rules that work and get rid the rules that don't work, and third step is generalization of the kept rules [2].

1.4 Conclusion

In this chapter the concept of animat problem and its components were introduced. It was shown that animat should perform adaptive behavior to survive and the control architecture has a central role toward this purpose. Different approaches to animat problem also were described and it was shown that one of the main approaches is the RL animat that the architecture of animat problem is matched with a reinforcement learning problem. For this project Wilson's animat that is a specific kind of RL animat is studied. The basis of Wilson's animat are similar to the original animat in [2] but the choice of environments and the algorithms of learning are more precise. There are many different LCS algorithms, but the most well-known and popular one is XCS classifier systems that is chosen and is studied in detail in Chapter 3. So, the purpose of this thesis is to solve and learn Wilson's animat problem to survive in different 2-D environments with several kinds of XCS classifier systems in different situations and scenarios. In the next chapter reinforcement learning and learning classifier systems are introduced.

CHAPITRE 2 REINFORCEMENT LEARNING AND LEARNING CLASSIFIER SYSTEMS

In the previous chapter the definition of animat problem and its structure were presented. It was stated that the adaptive behavior is essential for survival task. The adaptive behavior can be modeled by a reinforcement learning model that animat learns to survive by receiving payoff from the environment. The focus of this thesis is on Wilson's animat that is a specific class of reinforcement learning animat problems. To make a mathematical expression for the Wilson's animat problem in this section Reinforcement Learning (RL) and Learning Classifier Systems (LCS) frameworks are introduced.

2.1 Reinforcement learning

2.1.1 Markov chain

A Markov process is a stochastic process in which each state depends only on the previous state. Markov chain is a Markov process which has discrete and countable number of states and operates in discrete time. Suppose that X is a random variable and X_t is the value of random variable at time t. $S = \{s_1, s_2, \dots, s_n\}$ is a state space which is the values that X can take at discrete times. The random variable X_t is a Markov chain if:

$$\Pr\left(X_{t+1} = s_j \middle| X_0 = s_k, \dots, X_t = s_i\right) = \Pr\left(X_{t+1} = s_j \middle| X_t = s_i\right) \quad (2.1)$$

It shows that the next state of random variable (Markov chain) X_{t+1} only depends on the current state X_t . Markov chain is a chain starting with X_0 which is: $(X_0, X_1, \dots, X_n)$. A probability $p(i,j)$ is the probability of going from s_i to s_j by one step and called transition probability. A Markov chain can be expressed based on transition probabilities. The mathematical expression of transition probabilities is:

$$p(i,j) = \Pr\left(X_{t+1} = s_j \middle| X_t = s_i\right) \quad (2.2)$$

Let's denote $\pi_j(t) = \Pr\,(X_t = s_j)$ as the probability that the chain is in state j at time t and denote $\boldsymbol{\pi}(\mathrm{t}) = [\pi_1(t)\ \pi_2(t)\ \dots] = [\Pr(X_t = s_1)\ \Pr(X_t = s_2)\ \dots]$. The dimension of $\boldsymbol{\pi}(\mathrm{t})$ is the same as dimension S. The chain will start with $\boldsymbol{\pi}(0)$. All of the elements of $\boldsymbol{\pi}(0)$ are 0 except

one of them which the random variable is in that state. From Chapman-Kolmogrov equation we can write:

$$\pi_i(t+1) = \Pr(X_{t+1} = s_i) = \sum_k \Pr\left(X_{t+1} = s_i | X_t = s_k\right) \Pr(X_t = s_k)$$
$$= \sum_k p(k,i)\pi_k(t) \qquad (2.3)$$

The probability transition matrix is denoted by $\boldsymbol{P}$ that $i,j\ th$ elements are $p(i,j)$. On the other hand sum of the rows elements of $\boldsymbol{P}$ are one ($\sum_j p(i,j) = 1$). Hence, $\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t)\boldsymbol{P}$ and so $\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0)\boldsymbol{P}^t$.

n-step transition probability ${p_{ij}}^{(n)}$ is the probability of starting from state i and after n steps reaching to state j after n states.

$${p_{ij}}^{(n)} = \Pr\left(X_{t+n} = s_j \middle| X_t = s_i\right) \qquad (2.4)$$

Where ${p_{ij}}^{(n)}$ is the $ij\ th$ element of $\boldsymbol{P}^n$.

A Markov chain $(X_0, X_1, \dots)$ may reach a stationary distribution π^*, where the state and after that next states are independent of initial condition. So, we will have:

$$\pi^* = \pi^*\boldsymbol{P} \qquad (2.5)$$

π^* is left eigenvector associated with the eigenvalue $\lambda = 1$ of $\boldsymbol{P}$ [16].

2.1.2 Definition and basic architecture of reinforcement learning

Reinforcement learning is learning based on maximization of reward for agent that performs in an environment. The idea of reinforcement learning is inspired from study of the behaviour of animals from psychological point of view. Animals or human many times do a lot of works without receiving any reward to reach to a later reward at the end. So, reinforcement learning is based on this idea [17]. For example in foraging, an animal does a lot of actions in search for food and the obtained food is a reward, actually this is a distant reward. In reinforcement learning finding food has a positive reward and motions that consume energy have negative reward or punishment. Reinforcement learning builds a computational model of this type for complex behaviour of animals. In reinforcement learning the role of environment is important because the agent can't act only based on some pre-defined rules in a changing environment and it should

change its action adaptively. Applications of reinforcement learning are in robotics, animals' behaviour, games, control theory and finance.

2.1.2.1 Architecture of reinforcement learning

Figure 1-1simply shows the architecture of reinforcement learning:

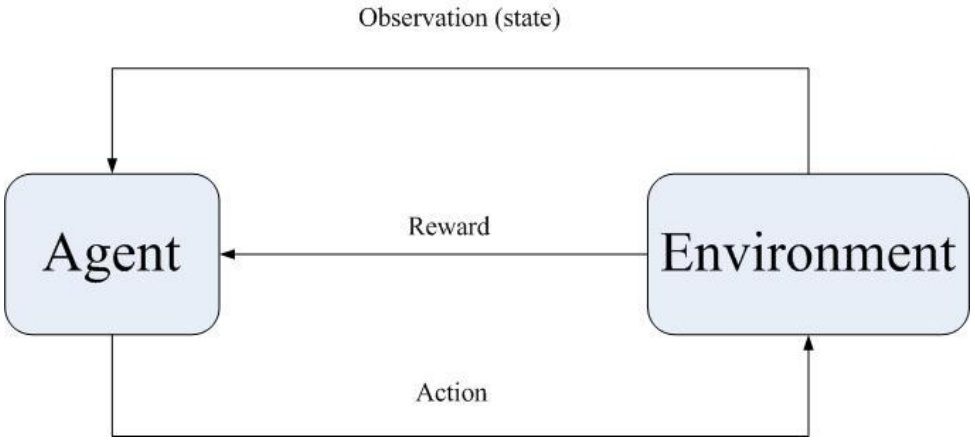


Figure 2-1: block diagram of a reinforcement learning problem.

In this diagram the agent first observes the environment that is the current state of the environment and then chooses an action and applies it to the environment. In the next step he receives an immediate reward from the environment for his action. The goal of agent is to maximize sum of the rewards. Agent should learn how to choose actions to obtain maximum sum of the rewards. It tries various actions in some states and after several times, learns which action is the best for which state. So, the agent in fact finds a policy (the rule of choosing an action at each state of the environment). There are methods in reinforcement learning which agent without predicting the effect of its action on the future rewards can learn optimal policy.

2.1.2.2 Problem statement

An agent in the environment, moves in discrete time steps denoted by t: $t = 0,1,2, \ldots$ and at each time step the agent observes the state of environment (that can be considered as the state of agent too) $s_t \in S$ where S is the set of possible states. According to the observed state, the agent

chooses an action $a_t \in A(s_t)$ where $A(s_t)$ is the set of possible actions that can be chosen at state s_t. In the next step $t+1$ the agent will receive reward $r_{t+1} \in R$ when it is in state s_{t+1}.

At each time step in each state, the agent chooses an action a_t from $A(s_t)$. It is a type of probabilistic mapping that is called policy and is denoted by $\pi_t(s,a)$. It represents the probability that $a_t = a$ if $s_t = s$. An agent tries to change the policy for the purpose of maximum return (total rewards) in long sense. The agent selects actions a_t to maximize the function:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \qquad 0 \leq \gamma < 1 \tag{2.6}$$

t is time step and the factor γ: $0 \leq \gamma < 1$ is discount factor which determines the importance of later and sooner rewards. For $T < \infty$ it is called episodic task.

2.1.2.3 Reinforcement learning in Markovian environments

The environment in which reinforcement learning tries to learn can be a Markovian environment or a non-Markovian environment with different levels of complexity for each one. For example, woods1 is a Markovian environment with eight obstacles and one food, woods101 is a non-Markovian maze environment with closed walls and low level of complexity and woods7 is a non-Markovian environment with a high variety of sensory patterns and high level of complexity (see Appendix 1). The number of similar cells in a non-Markovian environment determines its complexity. Actually, the environment is a problem that agent tries to solve. A Markovian environment in the architecture of reinforcement learning leads to a Markov decision process. This kind of reinforcement learning is called reinforcement learning in Markovian environments. A Markov decision process (MDP) satisfies:

$$\Pr(s_{t+1} = s', r_{t+1} = r | s_t, a_t) = \Pr(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0) \tag{2.7}$$

In fact, Markov decision process is the extension of Markov chain when action and rewards are considered. The probability space is the set of different states of the environment (e.g. sensory states).

To make a mathematical expression of a reinforcement learning problem in Markovian environments transition probability ${P^a}_{ss'}$ and expected value of the next reward $R^a_{ss'}$ are defined as follows:

$$P^{a}{}_{ss'} = \Pr(s_{t+1} = s' | s_t = s, a_t = a) \quad (2.8)$$

$$R^{a}_{ss'} = E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'] \quad (2.9)$$

Transition probability $P^{a}{}_{ss'}$ is the probability that the state changes from s to s' given action a. The expected value of the next reward $R^{a}_{ss'}$ is the average of receiving reward r_{t+1} in changing from state s to s' with action a. $P^{a}{}_{ss'}$, $R^{a}_{ss'}$ specify the dynamic of a finite MDP (MDP with finite number of states and actions).

(Note that the definitions of conditional probability and conditional expectation value are $\Pr(X|Y) = \frac{\Pr(X \cap Y)}{\Pr(Y)}$ and $E[X|Y = y] = \sum_{x \in \chi} x \Pr(X = x | Y = y)$.)

2.1.2.4 Policy

Policy is a mapping from state to action at each time step and is denoted by $\pi_t(s, a)$ that is probability of $a_t = a$ when $s_t = s$. The agent changes the policy to maximize the return in long sense. To represent change of the policy for the maximum return two functions can be used: *state-value function* and *action-value function*.

a) State-value function

State-value function $V^{\pi}(s)$ is the value of state s under policy π:

$$V^{\pi}(s) = E_{\pi}[R_t | s_t = s] = E_{\pi}\left[\left(\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1}\right) | \, s_t = s\right] \quad (2.10)$$

$V^{\pi}(s)$ can be written in a recursive form:

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \quad , a \in A(s), s' \in S \quad (2.11)$$

This equation is called Bellman equation and V^{π} is a unique solution for its Bellman equation [18].

To reach to the purpose of reinforcement learning (maximization of the return function) one should find a policy that maximizes the value function. In MDP, this policy is called optimal policy and is denoted by π^{*}. The optimal policy is not unique. The maximum state-value function is called optimal state-value function V^{*}.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad s \in S \tag{2.12}$$

b) Action-value function

Another useful function is $Q^{\pi}(s,a)$ which is the value of taking action a in state s under policy π:

$$Q^{\pi}(s,a) = E_{\pi}[R_t | s_t = s, a_t = a] = E_{\pi}\left[\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right) | \, s_t = s, a_t = a\right] \tag{2.13}$$

This optimal policy gives an optimal action-value function Q^*:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad s \in S, a \in A(s) \tag{2.14}$$

$Q^*(s,a)$ can be written in terms of $V^*(s)$:

$$Q^*(s,a) = E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] \tag{2.15}$$

The Bellman equation for $V^*(s)$ is called Bellman optimality equation and can be written as:

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V^*(s')] \tag{2.16}$$

So, the Bellman optimality equation for Q^* is:

$$Q^*(s,a) = \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma \max_{a'} Q^*(s',a')] \tag{2.17}$$

For finite MDP, Bellman optimality equation has a unique solution that is independent of policy. This solution is composed of N solutions according to N unknown states. If $P_{ss'}^{a}$, $R_{ss'}^{a}$ are available, the Bellman optimality equation can be solved for V^*, Q^*. The purpose in reinforcement learning is to find π^* to maximize V^{π} or Q^{π} [18].

There are at least two methods to solve this optimization problem: Dynamic programming and temporal difference learning. Dynamic programming is used for conditions when we know the model of environment i.e. the transition matrices and expected rewards. But temporal difference is used when we don't know transition matrices and expected rewards. So, temporal difference learning methods are more general and useful for higher variety of problems. In the next section we introduce temporal difference methods.

2.1.3 Temporal differences

In the situations that the transition matrices and expected rewards are not available, the agent can learn by interaction with the environment. At this situation temporal difference methods are used. The most well-known method in temporal differences is Q-learning.

Temporal differences follows a policy π to predict and update estimate of V^{π}. If state s_t at time t is observed, it updates the estimation of $V(s_t)$. Temporal differences method at time $t+1$ makes a target and updates according to the observed reward r_{t+1} and estimate $V(s_t)$:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2.18)$$

$r_{t+1} + \gamma V(s_{t+1})$ is called the *target*. The algorithm for temporal differences based on [18] is as follows:

- Initialize $V(s)$
- Repeat:
    - Initialize s
    - Repeat for each step :
        - $a \leftarrow$action that is given by π for s
        - The next state s', reward r,and action a are taken
        - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
        - $s \leftarrow s'$
    - End for the final state s
- End after enough iterations

2.1.3.1 Q-learning

Q-learning [19] is one of the most important developments in reinforcement learning. In its simplest form it is mentioned as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (2.19)$$

It directly approximates Q^* and always converges to the optimal value [20]. The optimal value in Q-learning is Q^* that remains unchanged (or with very small changes) after several iterations of the algorithm. The key to proof the convergence of the Q-learning is a Markovian process called the action replay process (ARP) [20]. For more details about the proof of convergence see [20]. The algorithm for Q-learning based on [18] is as follows:

- Initialize $Q(s, a)$
- Repeat:
    - Initialize s
    - Repeat for each step :
        - choose action a from s using a policy obtained from Q
        - The next state s', reward r,and action a are taken
        - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        - $s \leftarrow s'$
    - End for the final state s
- End after enough iterations

2.2 Learning Classifier Systems

2.2.1 Definition and Introduction

The world and the systems that it encompasses are composed of interconnected parts that as a whole function in a way different from the function of the individual parts. These complex systems are composed of interacting components. Complex adaptive systems (CAS) are complex systems with the capacity to learn from experience. CAS might be represented by a group of rule-based agents. Rules are in the form of "IF condition THEN action". These rules use the information from the environment to make decisions. The idea of LCS is evolving a population of rules that can collectively model a complex system. The system uses evolution to create new adaptable rules for the better operation of the system. The LCS algorithm outputs classifiers to

collectively model an intelligent decision maker. LCS employs learning to guide the evolution toward a better set of rules. Environment is the source of input data. LCS receives payoff by interaction with the environment. A learning classifier system learns to classify input messages from the environment and put them into general sets. Genetic algorithm is used in classifier systems to evolve rules and create new rules (evolution). Learning classifier system starts from random rules and learns and improves new rules. Learning classifier systems can solve reinforcement learning problems, classification problems, and function approximation problems. In LCS population of classifiers contains knowledge of the system [21].

2.2.2 How does LCS work?

The function of learning classifier system is to provide a set of condition-action rules that at each situation the agent can make its best decision for choosing action to obtain maximum total reward. It tries to achieve this goal by combining reinforcement learning techniques and genetic algorithm evolutionary approach. At the heart of the system is a set of rules that each rule has a parameter that can be increased when that rule receives reward from the environment. The environment at each state is represented in the form of a string for the system that can be matched by some rules in the population of rules. An auction among the matched classifiers determines the winner classifiers that their action can affect the environment. The reinforcement that is given by the environment updates the system for the next cycle. In this way the knowledge of system increases about the environment and the system is learned to operate in the environment. The genetic algorithm performs on the population of classifiers to generate new useful rules and increase the performance and generality of the system. The block diagram of a learning classifier system and its interaction with the environment is represented in Figure 2-2. A learning classifier system is composed of three components: rule and message subsystem, credit assignment subsystem, and classifiers discovery mechanism. Sensors, actuators, classifier population ([P]), and matching blocks are components of the rule and message subsystem, Auction, Payoff, and Taxes blocks are components of the credit assignment subsystem, and classifier discovery (GA) block is the main component of the classifier discovery mechanism [22].

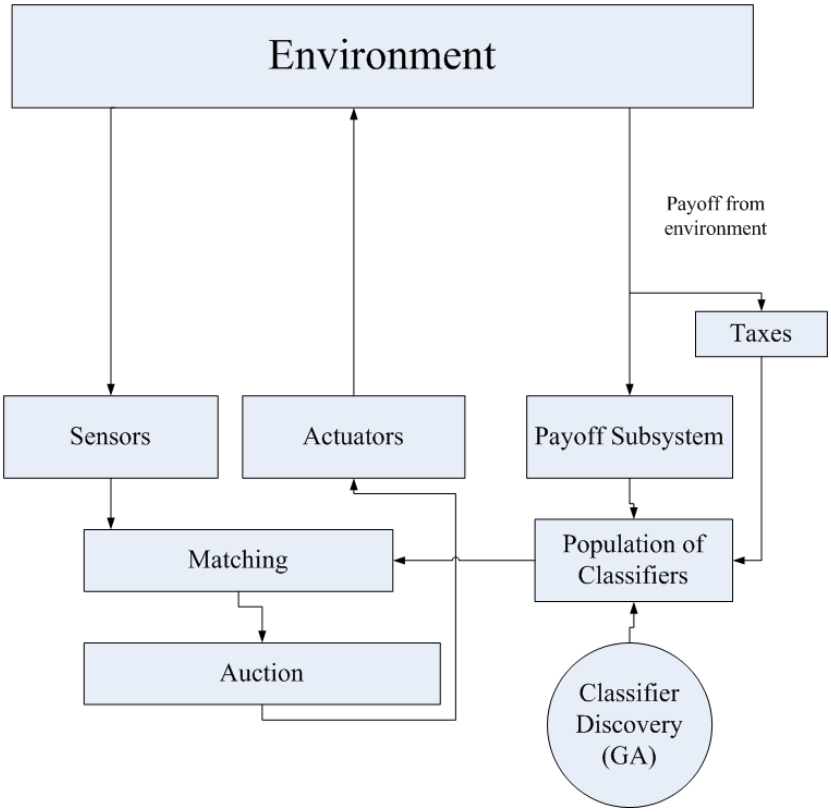


Figure 2-2: interaction of LCS with the environment [23].

2.2.2.1 Rule and Message subsystem

Each rule that is called "classifier" consists of a "condition" that is a word composed of ternary alphabet $(0,1,\#)$ and an "action" that is a string of (0s and 1s). The classifier is in this template:

$$IF < Condition > THEN < Action >$$

is called don't care which can be 0 or 1. This allows rules to be more general, i.e. the more #, the more general rule. This property can be measured by defining "specificity" of a classifier which is the number of non # symbols in the condition. For a rule with all # characters, the specificity is zero, and for a rule without # characters the specificity is equal to the length of the string. Rate $p_{\#}$ which is user dependent identifies the number of # in a classifier.

The set of actions depends on the type of the problem. For example in robotic, action can be "go left" or "go right", etc. In Wilson's animat problem, action can be one of 8 possible movements to different directions.

Classifiers compare with the messages from the environment and are tested to match or not. In the matching between condition and a message, every part of them should be matched. For example environmental message 011001 match with classifiers 0110#1, 01100#, ##100#, and ######. The classifier is matched with the message from the environment if the condition of classifier is matched with the condition of the message and the action of classifier is matched with the action of the message.

Each classifier has a portion which gives a measure for the rules' past performance in the environment. This portion is called the strength (fitness). A better performance of a classifier gives a higher strength. A classifier with higher strength when the condition matches an environmental message is more probable to reproduce when GA is applied because GA selects classifiers based on a probability proportional to their strength in the population.

The messages from the environment first enter to the sensor part of the classifier. Sensor block filters the message by selecting certain aspect of environment and then translates it to binary form to be processed by the classifier system. The actions of classifiers can perform on the environment by the actuators.

2.2.2.2 Credit assignment subsystem

In credit assignment subsystem, the classifier system learns by modification of the strength (fitness) according to the received reward from the environment. This modification process is composed of the three mechanisms: Auction, Payoff, and Taxes. A competition is held between classifiers that are matched with the environmental message in Auction block. In competition a bid is submitted in the auction. In the bid a winner classifier is selected to affect the environment. The reward or punishment (payoff) that environment gives, enters to the Payoff block to increase or decrease the strength of the winner classifier. At the end taxation performs on each classifier which submits a bid during the auction [22].

1. Auction

The classifiers that are matched with the environmental messages will be chosen and put in "match set" $[M]$. These classifiers go to the auction and each one submits a bid $B_i(t)$ to compete. The classifiers that have the highest bid will be copied in $[C]$ (set of winner classifiers) and are called winner classifiers. The total collective bids of $[C]$ are placed in B_{total}. Note that many

times $[C]$ has just one member because only one classifier can obtain the highest bid. It is the winner classifier. But it is possible to have two classifiers that both of them have the highest bid. In that case $[C]$ has more than one member. The classifiers in $[C]$ all have the same action. This action is sent to the actuators to perform on the environment. Based on that action, the environment gives a payoff in the next iteration.

The bid of classifier i at iteration t, is:

$$B_i(t) = k_0(k_1 + k_2 BR^{BRP})S_i(t) \quad (2.25)$$

k_0: Classifier bid coefficient. It is positive, constant and less than one. It acts as an overall risk factor.

k_1: Bid coefficient 1. It is constant and less than one.

k_2: Bid coefficient 2. It is constant and less than one.

$S_i(t)$: Strength of classifier i at iteration t.

BR: Measure of normalized specificity of classifier. $BR = 1$ if only one possible message matches each condition. $BR = 0$ if the condition consists of all # characters and classifier is matched by any message.

BRP: determines the importance of BR. Default value for BRP is 1.

2. Payoff: A well-known reinforcement algorithm is Bucket Brigade algorithm in which the strength is updated iteratively. In Bucket Brigade algorithm the environmental modification is beneficial or detrimental. For a beneficial modification the winner classifiers of auction receive a positive feedback and their strength increase and for a detrimental modification, they receive a punishment and their strength decrease. For each winner classifier i in $[C]$ a Payoff process is expressed as:

$$S_i(t+1) = S_i(t) + R_i(t) - B_i(t) + P_i(t) \quad (2.26)$$

Where $S_i(t)$ is the strength of the classifier i at the beginning of iteration t. $R_i(t)$ is the reward from environment during iteration t. $B_i(t)$ is the classifier's bid during iteration t. $P_i(t)$ is the total payments made to this classifier by $[C]$. $R_i(t) \neq 0$ for a winner classifier in auction on the previous iteration. Negative $R_i(t)$ means the punishment and positive $R_i(t)$ means the reward. The reward of action at iteration t will be applied at iteration $t+1$.

3. Taxes

Taxes are used to limit the strength of the classifier to be high or little strength. There are two types of taxes: life tax and bid tax.

Life tax: It is a type of tax with fixed rate that is applied to every classifier. Its aim is to reduce the strength of classifiers that rarely or never are matched and are not being used. Life tax decreases the strength of these classifiers and makes them candidate for replacement.

Bid tax: It is a type of tax with a fixed rate that is applied to each classifier which bids during an iteration. It penalizes general classifiers. General classifiers are the classifiers that bid on every step but never win because they have a low specificity which yields to low bid and makes a low chance for winning the auction.

Half-life that is the magnitude of the life tax is defined as

$$n = \frac{\log\left(\frac{1}{2}\right)}{\log\left(1 - Tax_{life}\right)}, \tag{2.27}$$

Where Tax_{life} is called tax rate.

After n iterations of inactivity (non-matching), the strength of an inactive (not matched) classifier would be

$$S(t+n) = S(t)\left(1 - Tax_{life}\right)^{n}. \tag{2.28}$$

So, the complete strength equation for the apportionment of credit mechanism will be

$$S_i(t+1) = \left(1 - Tax_{life}\right)S_i(t) + R_i(t) - B_i(t) - Tax_{bid}B_i(t) + P_i(t)\ . \tag{2.29}$$

2.2.2.3 Classifier discovery mechanism

Rule discovery is the process of introducing better rules (higher payoff) that doesn't exist in the population. A well-known mechanism for classifier discovery is genetic algorithm. It performs on

the population of classifiers by selecting one or two classifiers and evolving them by crossover and mutation.

2.2.2.3.1 Genetic algorithm

The genetic algorithm is a robust search algorithm based on the natural selection mechanism that adapts a population to the environment. In genetic algorithm, the genetic operators recombine the selected string (e.g. a bit string or condition part of a classifier) to make a new string for the next steps. The basic operators of genetic algorithm are selection, crossover, and mutation that perform consequently. The general algorithmic description of genetic algorithm based on [23] is as follows:

- Initialize parameters
- Make the initial population with initial fitness
- Repeat:
    - Selection of parents to produce offspring
    - Crossover
    - Mutation
    - Update population and the fitness of individuals
- End after enough iterations

Selection depends on the individual's fitness (strength). It uses the selection probability that is proportional to individual's strength. The higher strength has higher probability of being offspring. The probability that individual i is selected for mating is:

$$P_i = \frac{S_i}{\sum_{k=1}^{n} S_k} \tag{2.20}$$

S_i is the strength of member i, and n is the total number of members. This probability is assigned to each individual of the population based on its fitness value.

Crossover takes a part of each parent's string and combines them to make two offspring. If length of each string is L, a random number k is selected in the interval $(1, L-1)$. Then the place of

first k character of pairs is replaced with each other. For example, suppose that two parent strings (condition) A and B with length 7 are chosen from the population:

$$A = a_1 a_2 a_3 a_4 a_5 a_6 a_7 \quad (2.21)$$

$$B = b_1 b_2 b_3 b_4 b_5 b_6 b_7 \quad (2.22)$$

For $k = 4$, the resulting strings are two offspring A', B':

$$A' = b_1 b_2 b_3 b_4 a_5 a_6 a_7 \quad (2.23)$$

$$B' = a_1 a_2 a_3 a_4 b_5 b_6 b_7 \quad (2.24)$$

Mutation: mutation is used to make random changes into the population with low probability. In mutation one bit of string (condition) changes based on the following rules:

$$0\ to\ a\ 1\ or\ \#$$

$$1\ to\ a\ 0\ or\ \#$$

$$\#\ to\ a\ 0\ or\ 1$$

'#' symbol is the "don't care" symbol which can be 0 or 1. In learning classifier systems the genetic algorithm performs on the population of classifiers. Two classifiers are selected and copied from the population (action set in XCS) with a probability proportional to their fitness. The crossover operator performs on the copied classifiers from a randomly selected point. Then the mutation performs on the resulting classifiers. The average fitness of the selected classifiers is considered for the resulting classifiers and they will be copied into the population. The genetic algorithm in learning classifier systems produces classifiers with new conditions and new fitness values to be used for new sensory information and make general rules.

2.2.2.4 What is the difference between classifier in machine learning and classifier in learning classifier systems?

Classifier in machine learning and classifier in learning classifier systems in their nature do the same task based on generalization using some examples. In machine learning classification task performs by assigning a criterion to a set of data. The criterion must be general enough to be used for any unseen data to be predicted in true class. Classifier in learning classifier systems is a set of rules that in condition has some # symbols in the condition part. The set should be general

enough to predict the best action for any new state in the environment according to the data about the state, action, and payoff that has acquired from the environment.

2.3 Conclusion

In this chapter reinforcement learning (RL) and learning classifier systems (LCS) were introduced and it was mentioned that LCS can be used to solve a reinforcement learning problem. The Wilson's animat problem is a LCS-based animat problem and can be solved using the algorithm that was introduced in this chapter. Among classifier systems methods, XCS is the most well-known and the most popular one and is very general. It has the property of generalization and uses Q-learning for credit assignment problem [21]. The description of XCS will be presented in the next chapter (Chapter 3) and will be used for learning of animat in some Markovian environment.

CHAPITRE 3 XCS AND THE ANIMAT PROBLEM

3.1 XCS :eXtended Classifier Systems

3.1.1 Introduction

XCS was introduced to overcome unsatisfactory behavior and performance of classical LCS. In classical learning classifier systems (LCS) the strength is used both as the fitness in genetic algorithm selection and as the prediction of payoff in the system. The prediction of payoff that shows how much reward may be achieved from a certain action is used to represent the performance of the system. The fitness is used to represent the strength of a classifier to be selected for reproduction. However, the prediction of payoff is insufficient to be used as fitness for genetic algorithm because the GA removes classifiers with less reward than others that in turn removes low-predicting classifiers but well situated for its environmental niche [21]. XCS is a class of classifier systems that the prediction of payoff for each classifier is separated from the fitness. XCS has a prediction of payoff that is a different value from fitness for each classifier. The fitness is equal to a prediction of accuracy that is defined as an inverse function of the classifier's average prediction error. In addition to accuracy-based fitness, XCS uses niches genetic algorithm in which niches are defined as the match sets. Niches are a set of states of environment that each one is matched with nearly the same set of classifiers. Each niche (set of states) of environment results in different values for the expected payoff. Another important specification of XCS is standard tabular Q-learning that is used to tackle with the credit assignment problem. In fact, the credit assignment part and GA part are separated based on accuracy.

The above specifications of XCS lead to two important properties: first, the population of classifiers build an accurate and complete mapping $X \times A \rightarrow P$ from state and actions to predictions of payoff that can't be found in classical learning classifier systems. And second, XCS evolves maximally general classifiers (classifiers general enough that changing a 1 or 0 in the bits of their condition makes them inaccurate) that lead the system to reach to optimal performance. In fact, in XCS learning guides the evolution to create best set of rules that map

state-action values to the prediction of payoff and thus introduces an intelligent decision making system. For reinforcement learning problems that generalization is important XCS can be used because it has generalization property over states. By the above descriptions XCS is superior to the classical learning classifier systems.

Panmictic GA and Niche GA: In panmictic GA the probability of individuals in population have equal chance to be selected for generation of offspring. The panmictic GA is used in function optimization. In learning classifier systems GA should solve a multiple optimization problem; this is why niche GA is applied to XCS. In classical learning classifier systems as described in the previous chapter the GA was panmictic and therefore it was performed on the population of classifiers. So, the new classifiers were discovered based on the selected classifiers in the population set. Niche genetic algorithm is the extension of panmictic genetic algorithm to work for problems dealing with finding multiple and diverse solutions. A population of diverse individuals can be obtained by using niche GA. In XCS niches are a set of states of environment that each one is matched by nearly the same set of classifiers and are defined by the match sets. Niche GA in XCS is the performing of the genetic algorithm on the match set instead of the population set. Niche GA in XCS converges to a population of niches that covers a set of payoffs. In [24] the idea of executing GA on action set instead of match set was presented that yields improvement in the generalization capability of system. So, in this project the niche GA performs on action set.

The description of XCS is presented based on [1].

3.1.2 Description of XCS

A general description of XCS is presented in a structural form in Figure 3-1 that many details have been removed to show the basic operation better. The basic blocks are similar to the classical LCS in which matching between sensory information and the condition of classifiers in the population determines a smaller set of rules and an action is selected based on a particular strategy from the matched classifiers to affect the environment. The effect of action is turned back to the system by a payoff from the environment that updates the population of classifiers and increases the knowledge of system about its environment.

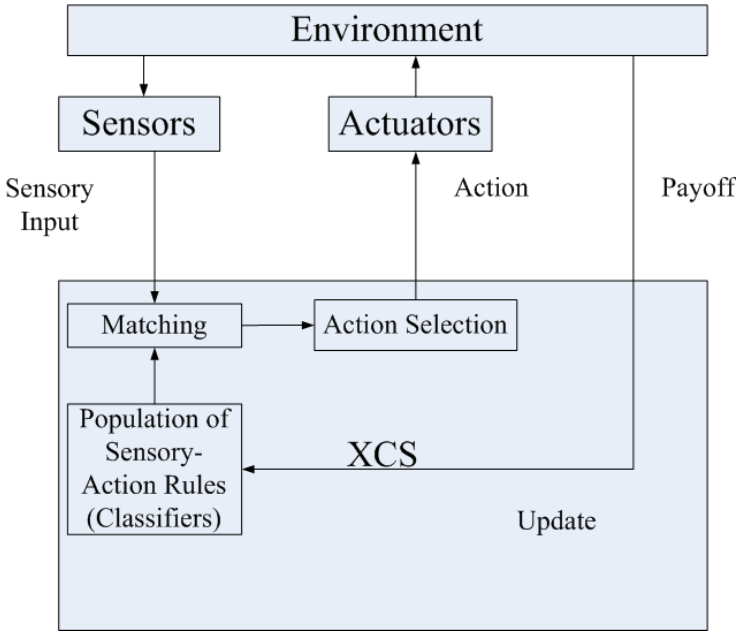


Figure 3-1: A general description of XCS.

Operation of XCS is illustrated in Figure 3-2 based on [24]. XCS interacts with the environment via sensors to receive sensory information, via actuators to perform action in the environment, and at each time step via a scalar delayed reinforcement (payoff) from the environment. In Figure 3-2 $[P]$ is the population set that contains the population of classifiers. Each classifier has two parts which are separated by ":", the left side is condition and the right side is action. Three values are associated with each classifier: p as the prediction, ε as the prediction error, and F as the fitness parameter. $[P]$ has a maximum size that is denoted by N. $[P]$ must be initialized at the start e.g. N classifiers that are generated randomly, or $[P]$ can be initialized empty. Initialization of p, ε, F can be arbitrary but usually are chosen around zero.

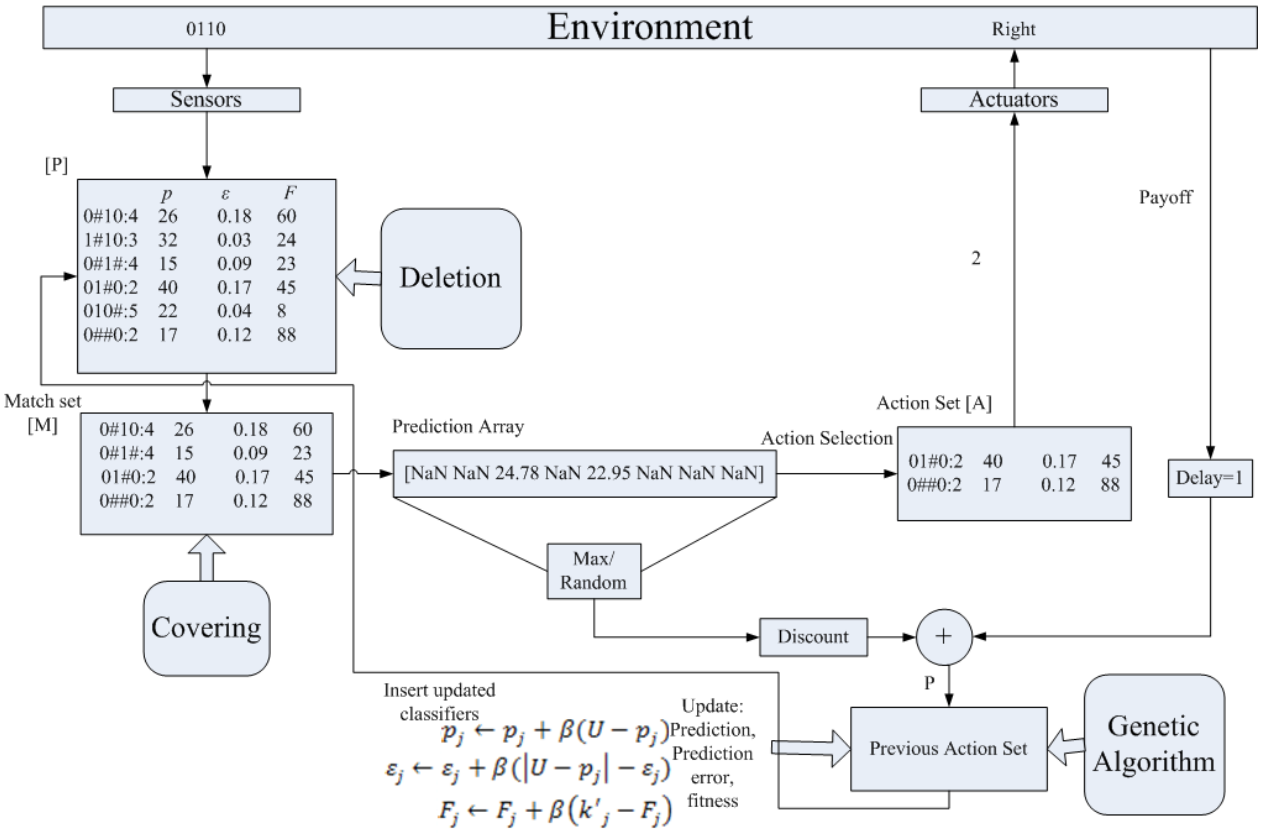


Figure 3-2: Detailed block diagram of XCS; inspired from [24].

3.1.2.1 Performance component

In this cycle each classifier in $[P]$ that its condition part matches with the sensory string, becomes a member of the match set $[M]$. Then a prediction array is constructed from match set by making system predictions $P(a_i)$ for each action a_i in $[M]$. $P(a_i)$ is equal to the weighted average of the predictions of classifiers that advocate a_i while weights are their corresponding fitnesses. So, the number of members in the prediction array is equal to the number of possible actions for the corresponding problem. If there is no classifier in match set for a possible action, the corresponding member of prediction array will receive NaN that means "no value". The classifiers advocating action with maximum $P(a_i)$ are transferred into action set $[A]$ (deterministic action selection) or the action is chosen completely random and the classifiers advocating that action are transferred into action set $[A]$ (random action selection). Then this action is sent to the actuators to perform action in the environment and an immediate reward r_{imm} is returned by the environment.

3.1.2.2 Reinforcement component

It deals with updating p, ε, F of classifiers in $[A]_{-1}$ that is the action set of the previous time step. To update p, ε, F standard Q-learning is used. This update is implemented by adding the discounted maximum of $P(a_i)$ of the prediction array (by multiplying discount factor γ ($0 < \gamma \leq 1$) to $maxP(a_i)$) and the previous time step external reward. Actually, it is $U = \gamma \max P(a_i) + r_{imm_{-1}}$. U is used to adjust the p_j, ε_j, F_j of the classifiers in $[A]_{-1}$ with learning parameter β ($0 < \beta \leq 1$); updating process for classifiers in $[A]_{-1}$ is as follows:

1. p_j is adjusted as $p_j \leftarrow p_j + \beta(U - p_j)$
2. ε_j is adjusted using U and the value p_j: $\varepsilon_j \leftarrow \varepsilon_j + \beta(|U - p_j| - \varepsilon_j)$, and finally,
3. Calculating F_j using the value of ε_j according to the method described later in section 3.1.2.4.

The Widrow-Hoff procedure ($p_j \leftarrow p_j + \beta(U - p_j)$ and $\varepsilon_j \leftarrow \varepsilon_j + \beta(|U - p_j| - \varepsilon_j)$ and the similar adjustment of F) is used after passing $1/\beta$ times update for a classifier. (note that j represents the involving classifier). Before $1/\beta$ times, updating procedure for each case is average of the previous values and the current one. To implement it, an "experience (exp)" parameter should be considered for each classifier showing number of updates (it is incremented every time the classifier enrolls in $[A]$). Using this kind of updating mechanism is called MAM technique. In a multistep problems that more than one step is needed to reach to a reward if at the start only one step is needed to finish the problem (the food is found within one step in animat case), the updates occur in $[A]$ and U is just current reward $U = r_{imm}$.

3.1.2.3 Discovery component

The GA acts on the action set $[A]$ and $[A]_{-1}$. The GA chooses two classifiers from $[A]$ (or $[A]_{-1}$) with probability proportional to their fitness. Then, it copies these two and performs crossover with probability χ on the copies, and performs mutation with probability μ per allele on them. Then if $[P]$ contains N classifiers (sum of numerosities of macroclassifiers. See 3.1.2.5) or more, two of them will be deleted stochastically from $[P]$ to make room. If $[P]$ has less than N classifiers, the copies are inserted into $[P]$ without deletion from $[P]$.

The deletion procedure is used to remove the low fitness classifiers from the population and keep approximately equal number of classifiers in each action set or environmental niche. The method used for selecting the classifiers that should be deleted is as follows:

A classifier is selected to be deleted by roulette-wheel selection. The deletion probability of each classifier is proportional to the action set size estimate of that classifier (a_s). The action set size estimate of each classifier is updated when that classifier enrolls in $[A]$ (or $[A]_{-1}$). To implement the deletion procedure a value "vote" is defined based on the action set size estimate. The algorithm for deletion is as follows:

$cl.x$ denotes one of the attributes of a classifier cl such as "condition", "action", etc.

```
Deletion ([P]):
  If "sum of fitness of classifiers in [P]" > N
      FPN ← "sum of fitness of classifiers in [P]" / "sum of numerosities of classifiers in [P]"
      Sumofvotes ← 0
      for each classifier cl in [P]
          vote ← cl.as * cl.n
          if cl.exp > θ_Del && cl.F/cl.n < δ * FPN
              then vote ← vote*FPN/(cl.F/cl.n)
          endif
          Sumofvotes ← Sumofvotes + vote
      endfor
      point ← rand (1) * Sumofvotes
      Sumofvotes ← 0
      for each classifier cl in [P]
          vote ← cl.as * cl.n
          if cl.exp > θ_Del && cl.F/cl.n < δ * FPN
              vote ← vote*FPN/(cl.F/cl.n)
          endif
          Sumofvotes ← Sumofvotes + vote
```

```
        if (Sumofvotes > point)
            if cl.n > 1
              cl.n = cl.n − 1
            else
              remove cl from [P]
            endif
        endif
    endfor
```

The rate of executing of genetic algorithm should be controlled. The reason is to assign the same number of classifiers to different match sets (niches) and make a complete mapping. Depending on the environment some match sets (niches) may occur more than others. The genetic algorithm performs in an action set if number of time steps starting from the last genetic algorithm in that action set becomes more than a threshold. To implement it, a counter is considered for each classifier when it is created. When action set is created, the average number of time steps is compared with the current counter (actual time (t_s)) and if their difference exceeds a threshold θ_{GA}, the GA performs on $[A]$ (or $[A]_{-1}$).

The discovery component includes also a covering mechanism. It is used when:

1. If there is no classifier to match with the environmental input. In this situation a classifier that its condition is matched with the input from environment and with the randomly chosen action is created to be inserted in $[P]$ and a classifier is deleted from $[P]$ using GA deletion method. After this process $[M]$ is formed.
2. System uses covering mechanism as an escaping method such as when it has stuck in a loop and go back and forward between two positions of the environment. In this situation creation of new classifiers that are matched can break the loop and if not, another covering will perform, and so on. Covering is needed at the starting of a run.

The execution of GA on action set lead to generation of a population with high fitness classifiers. These high fitness classifiers build a complete mapping of $X \times A$ space. In XCS defining fitness based on accuracy makes a better performance and yields the generalization ability. Niche GA leads to accurate and maximally general classifiers (classifiers with low error and general enough that changing one bit of 1 or 0 to # makes it inaccurate). Note that if a classifier with action a has

an accurate and maximally general condition, another classifier with the same condition but with different action is not in general accurate and maximally general.

3.1.2.4 The fitness calculation

A fitness is updated when it enrolls in $[A]_{-1}$. It is updated by a value which depends on the accuracy of classifier. This accuracy is relative to the other accuracies of classifiers in the set. This calculation has three steps:

1. Calculate classifier's accuracy k_j which is function of current value of ε_j:

$$k_j = \begin{cases} \alpha \left(\frac{\varepsilon_j}{\varepsilon_0}\right)^{-\nu} & \varepsilon_j > \varepsilon_0 \\ 1 & otherwise \end{cases} \quad (3.1)$$

   Note that $0 < \alpha < 1$. So, k_j is decreasing function for $\varepsilon_j > \varepsilon_0$.

2. Calculating relative accuracy k'_j: for each classifier, k'_j is obtained by dividing its accuracy by the total of the accuracies in the set.
3. Adjusting the fitness of classifier F_j: before $\frac{1}{\beta}$ times adjusting of F_j, F_j is set to the average of the current k'_j and previous values of k'_j. But after $\frac{1}{\beta}$ times adjusting of F_j,

$$F_j \leftarrow F_j + \beta\left(k'_j - F_j\right) \quad (3.2)$$

3.1.2.5 Macroclassifiers

For each classifier in the population a numerosity (n) component is considered. When XCS generates a new classifier population of classifiers is checked out to see if any classifier with the same condition and action of the new generated classifier is available. If no, the new classifier is added to the population with its own numerosity that is set to one. But if yes, the classifier is not added to the population and one is added to the numerosity of classifier. These classifiers are called macroclassifiers. One macroclassifier with numerosity n, is equivalent to n classifiers. All the functions of XCS work sensitive to numerosity. For example in calculating k'_j, a macroclassifier with numerosity n behaves such as n separate classifiers.

3.1.2.6 List of parameters

$\boldsymbol{N}$ the maximum size of the Population (sum of the numerosities of classifiers).

$\boldsymbol{p_I, \varepsilon_I, F_I}$ the initial prediction, prediction error, and fitness of each classifier in the population. They should be initialized to a positive value around zero.

$\boldsymbol{\beta}$ the learning rate of p, ε, F. It is usually set to beta=0.1-0.2.

$\boldsymbol{\gamma}$ discount factor

$\boldsymbol{\varepsilon_0, \alpha}$ parameters of the accuracy function. α is usually set to 0.1 and ε_0 is usually set to 10 for animat problem in some 2-D environments.

$\boldsymbol{\nu}$ is used in calculation of fitness and is usually set to 5.

$\boldsymbol{\theta_{GA}}$ GA threshold which determines whether performing GA on $[A]$ (or $[A]_{-1}$) or not. When the average time since the last GA in the last action set is greater than θ_{GA}, GA happens in a set (θ_{GA}=25-50)

$\boldsymbol{\chi}$ probability of applying crossover in the GA. (χ =0.5-1)

$\boldsymbol{\mu}$ probability of mutating an allele in the offspring. (μ =0.01-0.05)

$\boldsymbol{P_\#}$ is the probability of one # in a place in the condition of a classifier. It is usually set to 0.33.

$\boldsymbol{\theta_{del}}$deletion threshold. The fitness of a classifier is considered in deletion probability, if the experience of a classifier (exp) is greater than θ_{del}. It is set around 20.

$\boldsymbol{\delta}$ the fitness of a classifier is considered in deletion probability if the fraction of the average fitness of [P] is less than δ. It is set to 0.1.

$\boldsymbol{\theta_{sub}}$ subsumption thresold. If experience of a classifier is larger than θ_{sub} it can subsume with another classifier. It is set to around 20.

$\boldsymbol{p_{explr}}$ probability of using random action selection. It is set around 0.5.

$\boldsymbol{\theta_{mna}}$ represents the minimum number of actions that have to be available in a match set [M]. If less minimum number of actions in a match set is than θ_{mna}, covering occurs. It is set to 8.

doGASubSumption; This parameter can be 0 or 1. If it is 1, GA subsumption occurs, and if it is 0, the GA subsumption doesn't occur.

doActionSetSubsumption; This parameter can be 0 or 1. If it is 1, action set subsumption occurs, and if it is 0, the action set subsumption doesn't occur.

3.1.3 Generalization

Generalization: Generalization is a property that different situations in the environment with equal consequences are recognized with lower complexity than the raw environmental data. In LCS generalization means that a classifier can match more than one input vector of the environment.

XCS forms a complete mapping $X \times A \Rightarrow P$ from state and action to the payoff prediction which tells that if at state x the action a is performed what would be the payoff. In XCS combination of accuracy-based fitness and niche GA leads to accurate and maximally general classifiers. Accurate classifier is a classifier with error less than ε_0. Maximally general classifier is a classifier that changing any 1 or 0 in its condition to # makes it inaccurate. The niches of the environment that have the same payoff but have different sensory inputs -that have been obtained by the evolution of generalized classifiers- merge to the same niche. In fact, this is the goal of XCS to put same payoff niches in one class (one niche). So, the resulting population will contain minimum number of separate conditions.

To describe the mechanism of the above hypothesis consider two classifiers $C1$, $C2$ where they have the same action and the condition of $C2$ is more general than $C1$ which means that condition of $C2$ can be obtained by changing one or more of 0 or 1s of condition of $C1$ to #. Suppose that $C1$ and $C2$ have the same ε. When $C1$ and $C2$ are in an action set, their adjusted fitness value is the same for both of them. However, since $C2$ is more general than $C1$, the probability that $C2$ occurs in more match sets is higher. In addition, since GA performs on the action set, the probability that $C2$ reproduces is higher. In the case that $C1$, $C2$ occur in the same action set, the exemplars of $C2$ will receive more fitness adjusted value. As the result more general classifiers will appear.

Subsumption: Subsumption is a technique that classifiers subsume with an existing accurate classifier and a group of subsuming classifiers replace the subsumed ones. So, the number of

classifiers is reduced. Actually, in subsumption we try to omit accurate but unnecessarily specialized classifiers. As an example suppose two accurate classifiers $C1 = 11\#\#:3$ and $C2 = 1\#\#\#:3$. $C2$ is more general than $C1$ and so, subsumption should omit $C1$ from the population [24]. For XCS two kinds of subsumption procedure often are used exist that introduced: action set subsumption and GA subsumption.

Action set subsumption: This procedure searches action set to find the most general classifier (accurate and sufficiently experienced) and the other classifiers subsume with it. The failed classifiers are removed and the winners (subsummers) are kept. The winner classifier and the failed classifiers must have the same action but the winner has to be more general than.

GA subsumption: When GA generates offspring, parents are examined to see if one or both of them are accurate and more general than offspring (parent subsume offsprig). If it occurs, the offspring is not added to the population and the numerosity of parent is incremented by one.

3.1.4 What are the applications of XCS?

XCS is a learning algorithm that can tackle to large variety of problems. It can solve the animat problem that is an environment navigation problem and is our goal in this thesis. However, it also can solve other problems [21] that have been listed below:

- Multiplexer function
- Real-valued Multiplexer problem
- Integer-valued data mining
- Function approximation
- Blocks world problem
- Rule-set reduction
- Distributed data mining
- Epidemic data mining

As an example XCS has the priority to the Q-learning method in solving animat problem (see 4.5).

3.2 Animat problem and 2-D environments

3.2.1 Multi-step problems

Reinforcement learning is problem of exploration and exploitation in the environment with distributed reward, and learning by performing one or more types of action selection strategies to maximize the total reward. Multistep problem is a reinforcement learning problem that the current sensory input depends at least to the previous time step action and the previous sensory input and at each time step the system may receive a reward. Wilson's animat is a multi-step problem.

3.2.2 Wilson's animat problem and 2-D environments

The animat problem that has been considered in this thesis is the Wilson's animat problem in 2-D environments. Wilson's animat is a multi-step problem in which an agent in a two dimensional rectangular maze environment continued toroidally at the edges learns to find a food. Some examples of maze environments in the literature of Wilson's animat are woods1, woods2, woods7, maze4, maze5, maze6, woods14, woods101, woods101 ½, and woods102. Woods1, woods2, maze4, maze5, maze6, and woods14 are Markovian environments with delayed reward i.e. the next input y (and the reward) only depends on the current input x and the current action a. Woods7, woods101, woods101 ½, and woods102 are non-Markovian environments that animat needs more history (memory) to decide about the next step. In Markovian environments with delayed reward, it is possible to use Q-learning for learning an optimal policy. Q-learning procedure for Markovian environments after enough iteration for every input, converges to a function $Q(x, a)$. However, Q-learning doesn't have the generalization ability and is not appropriate for big and complex problems. This is why XCS combines the generalization ability of LCS with convergence ability of Q-learning and introduces a learning algorithm that can work for Markovian environments.

Each environment contains foods and obstacles. For each position in the environment a payoff is considered for example food has a positive reward. The animat is equipped with eight sensors around it and has actuators to move toward eight different directions by one step at each move. Each object in the environment has a sensory code. Animat senses the eight surrounding cells and builds a detector vector composed of consequence of sensory codes starting from north cell and

in clockwise direction. The action is a one step move toward a neighboring cells numbering from 0 to 7 and starting from north cell in clockwise direction. For a blank cell animat can simply move toward it. If there is a rock in the cell that animat decides to move, the move is not permitted to take place but one time-step passes. If the cell is food, the animat moves to the cell and receives a reward (usually 1000 but can be any positive value without any difference in performance). There are several environments that are used to test the animat problem. Woods1, woods2, maze5, and also S2DM (see Chapter 5) and Complex environments (see Chapter 5) are used in this thesis to test the assumed animat problem. In woods1, maze5, and also S2DM and Complex environments the sensory code for foods ("F") are 11, for rocks ("O") are 10, and for blank cells (".") are 00. The sensory string is a 16 bits string. Woods2 is a more challenging version of woods1 environment. The environment has two types of objects each with two different kinds: "F" and "J" are two kinds of food with sensory codes 110 and 111 and "O" and "K" are two kinds of rock with sensory codes 010 and 011. Blanks "." have sensory code 000. The sensory codes can be anything else but have to be different to identify a different object. A meaning also can be given to the codes but it makes no difference; for example the codes in woods2 can be thought like this: bit 0 for smell (1 tastes good and 0 doesn't have taste), bit 1 for solidness (1 solid and 0 not solid), and bit 2 for color (1 red and 0 blue). The sensory string is a 24 bits string. Maze5 is a more complex environment and learning in it is more difficult. It contains 36 blank cells, 44 obstacle cells and only one food cell. The environments woods1, woods2, and maze5, are illustrated in Figure 3-3 to Figure 3-5. The goal of Wilson's animat is learning to find a food as fast as possible. The optimal performance that is the average of numbers of steps to food starting from any blank point of the environment is a constant value for each environment. For woods1 and woods2 it is around 1.7 steps to food, and for maze5 it is around 5 steps to food.

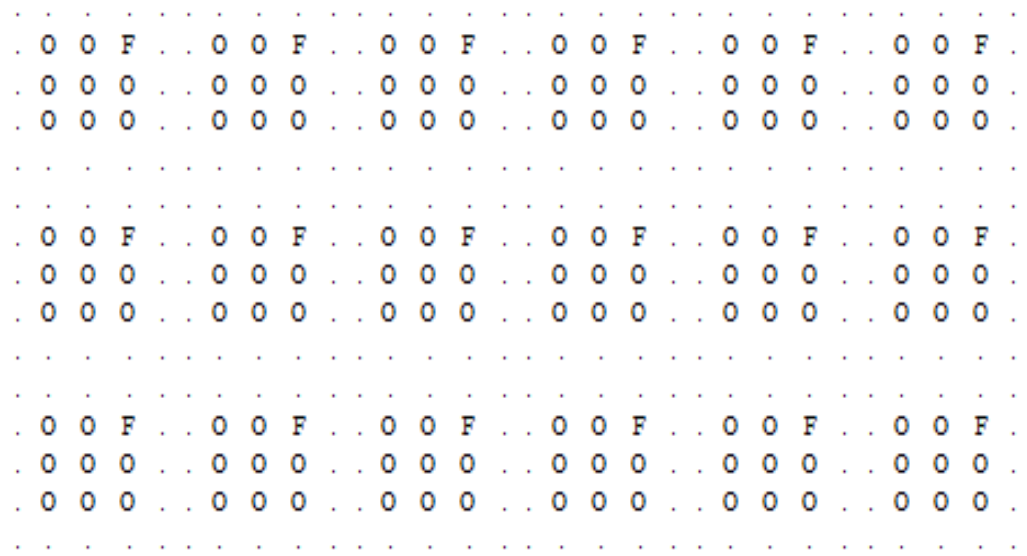


Figure 3-3: The environment Woods1; inspired from [25]

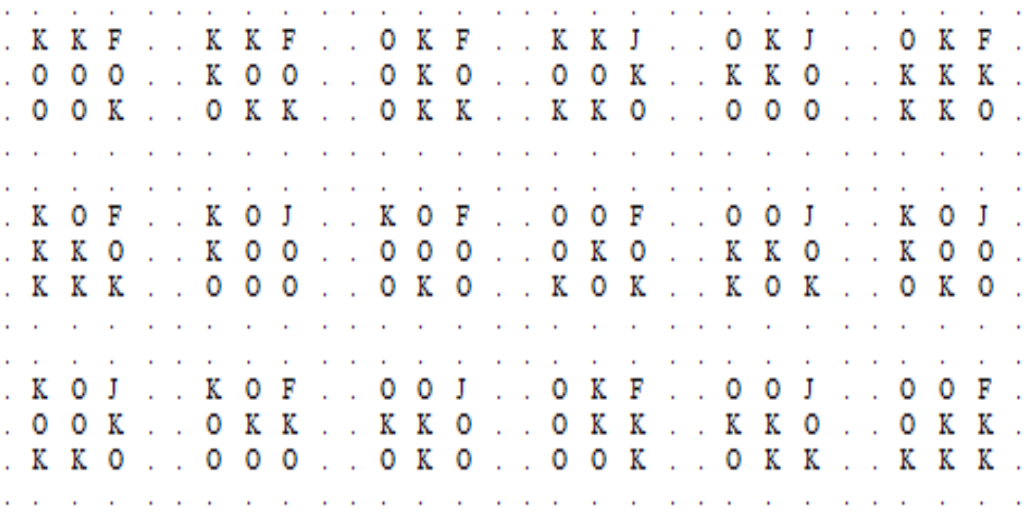


Figure 3-4: The environment Woods2; inspired from [1]

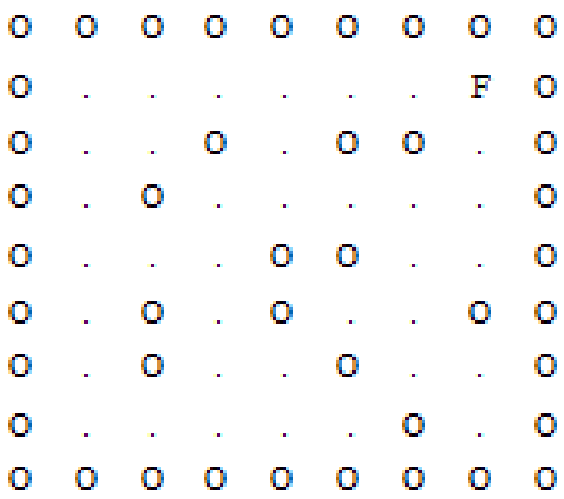


Figure 3-5: maze5 environment; inspired from [26]

3.3 XCS animat problem

In LCS literature, XCS is the most well-known and has the superiority to other LCS methods and has the ability of generalization. This is the reason that Wilson solved the animat problem in non-Markovian environments (woods2) using XCS. In this thesis this approach is considered to deal with the animat problem. So, control architecture of animat in this way is a XCS algorithm. Animat tries to find a food in a two dimensional Markovian 2-D environment with delayed reward that at each step takes the sensory information and based on XCS performs an action and based on that action and its current location in the environment receives a reward in the next step. The goal is to build a map $X \times A \Rightarrow P$ with a population of accurate and maximally general classifiers that help animat to find food easily. The block diagram of XCS animat is illustrated in Figure 3-6.

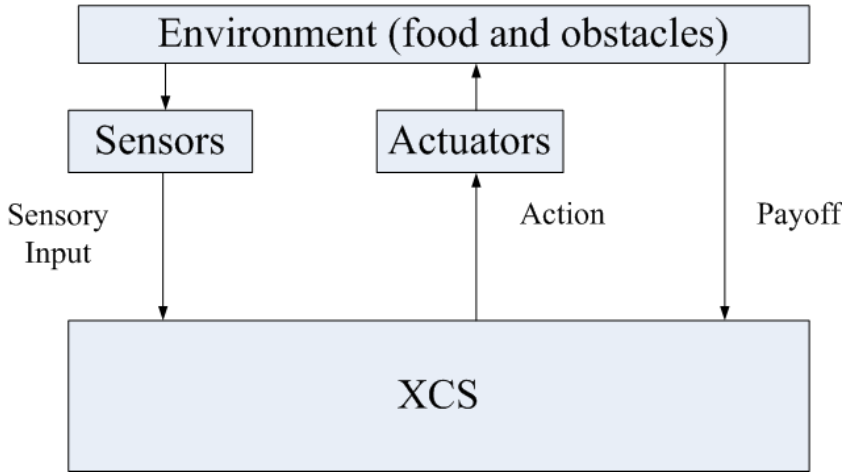


Figure 3-6: Block diagram of a XCS animat

3.4 Experiment

The Wilson's animat is considered as the animat problem and the XCS algorithm is considered as the solution for this problem. The Wilson's animat tries to search environment for food. Only moving random steps may reach the animat to food, but it is not an intelligent (!?) animat to move only randomly. The animat must learn to find food as fast as possible when it starts from a random point in the environment. For this purpose the animat should build a mapping of environment in its mind. XCS is the brain of the animat and the final population of classifiers is its learned mind that helps it to find food as fast as possible. The final population provides the animat mapping $X \times A \Rightarrow P$ in a compact (generalized) form. Animat obtains this population by searching in the environment and exploration of data that tells it the payoff available in each state and exploitation and generalization to tell it which action is the best at each state.

3.4.1 Experimental setting

The experiment that is living of animat in a two-dimensional environment is performing of several "problems" iteratively. Each problem is putting animat in a blank cell that is selected randomly and then moving under control of XCS system that has the role of brain for animat, until a food is reached. At that point, a new problem from a new random blank cell begins.

Action selection regime: At the start of each problem, XCS decides to use randomly action selection (explore) or deterministic action selection (exploit). Exploration is choosing action randomly and exploitation is choosing action that its prediction payoff is the highest (deterministic). In exploration mode GA operates in normal mode, covering and reinforcement components occur, and actions are selected randomly from actions that their prediction payoffs in the prediction array are non-zero. In exploit mode GA is turned off but covering mechanism works; updates for $[A]_{-1}$ works but not for $[A]$; and in performance component the actions with maximum prediction in the prediction array are chosen.

The initial population is initialized empty. The population of classifiers is updated from problem to problem and is not initialized when a problem starts. The performance is measured as the average steps to food for the last 50 exploitation problems.

Although in the basic framework of XCS a classifier is defined by five components (condition, action, prediction, prediction error, and fitness), but in the experiment more components are needed to define a classifier. A classifier is implemented with nine parts in the experiment with XCS for animat problem. The components of a classifier are: condition, action, performance, performance error, fitness, experience, time stamp, action set size estimation, and numerosity. These nine parts are variables of a classifier that make a classifier as an object.

"Condition" and "action" identify the essence of a classifier, i.e. if two classifiers have the same "condition" and "action" but with difference in the other components; they are assumed as one classifier in different time steps. "Condition" is used as a component of a classifier to communicate with the sensory information. On the other hand "action" is used to interact with the environment by doing modification in the environment. It is in fact the motor system of the animat. "Prediction (p)", "prediction error (ε)", and "fitness (F)" are used to give a value to classifier for application in the reinforcement cycle of XCS. "Experience (exp)" component identifies the number of times that a classifier is enrolled in the action set. Each time that a classifier enrolls in the action set the "experience" it increases by one. "Actual time (t_s)" is a number that identifies the last time that the genetic algorithm has occurred in the action set and contained this classifier. It is used in GA sub-cycle to introduce a condition for running of GA sub-cycle. "Action set size estimation (a_s)" is a value that represents the average size of action

sets that the corresponding classifier has enrolled. This value is used in the Deletion sub-cycle. "Numerosity (n)" indicates the number of microclassifiers that are represented by this classifier.

In multi-step problems that a problem finishes in more than one step, updates of prediction, prediction error, and fitness perform on $[A]_{-1}$ using definition of U as: $U = \rho_{-1} + \gamma maxP(a_i)$ where ρ_{-1} is the previous step reward. But in single step problems that it is finished by moving one step, the update operates on $[A]$ using definition of U as: $U = \rho$ where ρ is the immediate reward.

The consequence of updates that are considered in [27] are $exp, p, \varepsilon, a_s, F$. Update of F requires calculating accuracy that makes its calculation more complex than the others. The mathematical expression for calculating the accuracy is $\alpha\left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu}$ for $\varepsilon \geq \varepsilon_0$ and 1 otherwise. Furthermore, the MAM technique is not used for updating of F. The GA also occurs in action set. In this algorithm the population of classifiers is a population of macroclassifiers that each one has a numerosity which represents the number of classifiers it contains. The covering mechanism assures the availability of certain number of actions in each match set.

The parameters of the system can be classified into five groups:

- Variables that have to be initialized: $N, p_I, \varepsilon_I, F_I$
- Parameters of the reinforcement cycle and fitness calculation: $\beta, \varepsilon_0, \alpha, \nu, \gamma$
- Parameters of the discovery component (GA and covering): $\theta_{GA}, \chi, \mu, P_{\#}, \theta_{del}, \delta, \theta_{mna}$
- Parameter of subsumption mechanism: θ_{sub}
- Parameters of the experiment: p_{explr}

3.4.2 Results for XCS animat in various two-dimensional environments

3.4.2.1 Developing XCS framework of animat problem in MATLAB

In this thesis MATLAB is used to implement XCS for the animat problem and for implementation the algorithmic description of XCS is used based on [27]. The presented algorithmic description is close to the original work with some changes according to the papers that have been published after that. The action selection strategy is a combination of exploration and exploitation that they alternate from cycle to cycle.

The following sections represent the results of learning of animat in different environments. The results are divided into "with subsumption" and "without subsumption" that the application of sebsumption mechanism is implemented by $doGASubsumption = 1$ and $doActionSetSubsumption = 1$. Three environments woods1, woods2, and maze5 are selected to test the learning of XCS animat. The numbers of 10000 problems are run for animat problem in each run of experiment.

The parameter setting for experiment with both XCS animat problem without subsumption and XCS animat problem with subsumption are represented in Table 3.1. The parameters are selected based on [27].

Table 3.1: Parameter setting for XCS without subsumption and XCS with subsumption

	N	p_I	ε_I	F_I	β	γ	ε_0	α	ν	θ_{GA}	χ	μ	$P_\#$	θ_{del}	δ	θ_{mna}	θ_{sub}	p_{explr}
Woods1	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Woods2	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Maze5	2500	0.0001	0.00001	0.001	0.2	0.71	5	0.1	5	25	0.8	0.01	0.3	20	0.1	8	20	1

3.4.2.2 Without Subsumption

Results of the XCS algorithm without subsumption in woods1, woods2, and maze5 are presented in Figure 3-7 to Figure 3-9.

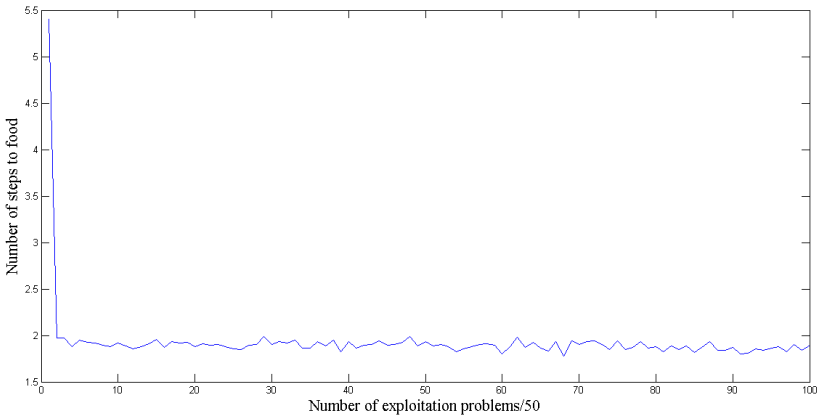


Figure 3-7: XCS animat in woods1 without subsumption, see Figure 3-3.

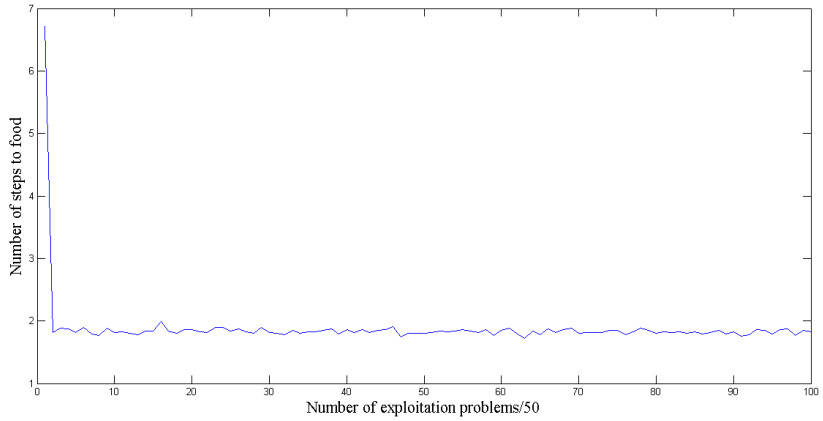


Figure 3-8: XCS animat in woods2 without subsumption, see Figure 3-4.

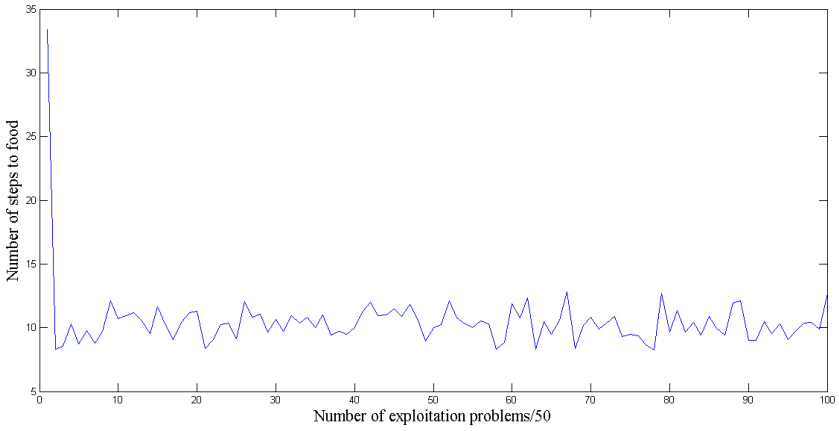


Figure 3-9: XCS animat in maze5 without subsumption, see Figure 3-5.

3.4.2.3 With Subsumption

Results of the XCS algorithm with subsumption in woods1, woods2, and maze5 are presented in Figure 3-10 to Figure 3-12.

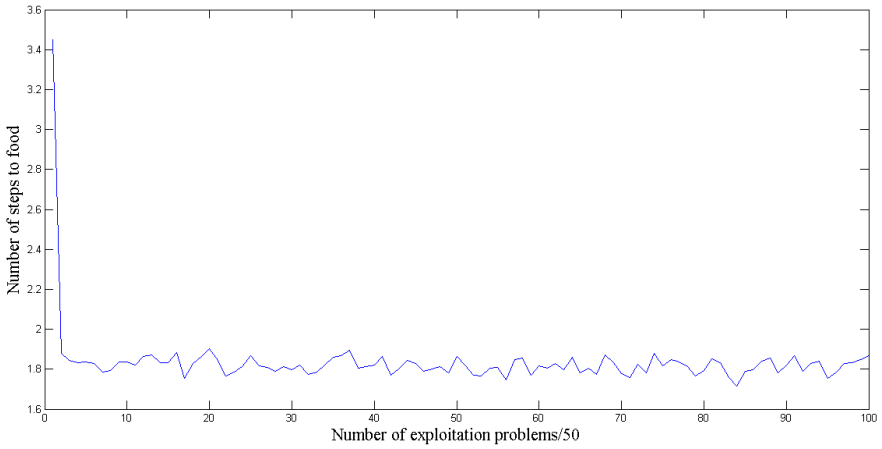


Figure 3-10: XCS animat in woods1 with subsumption.

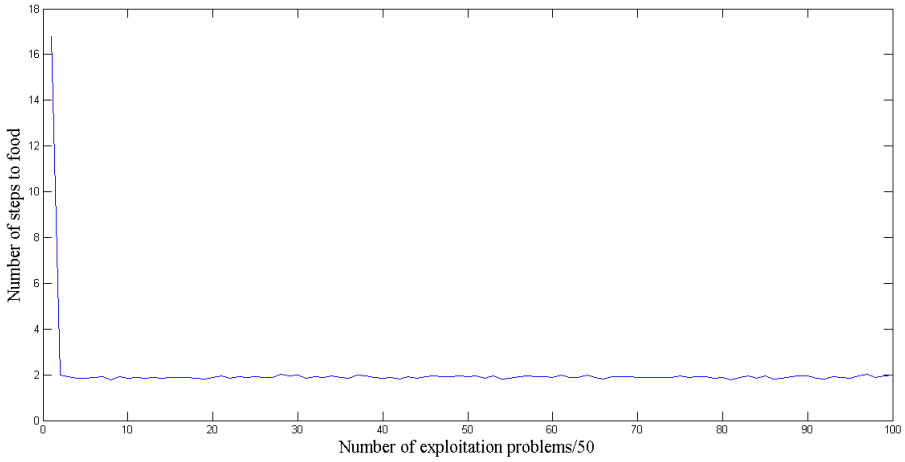


Figure 3-11: XCS animat in woods2 with subsumption.

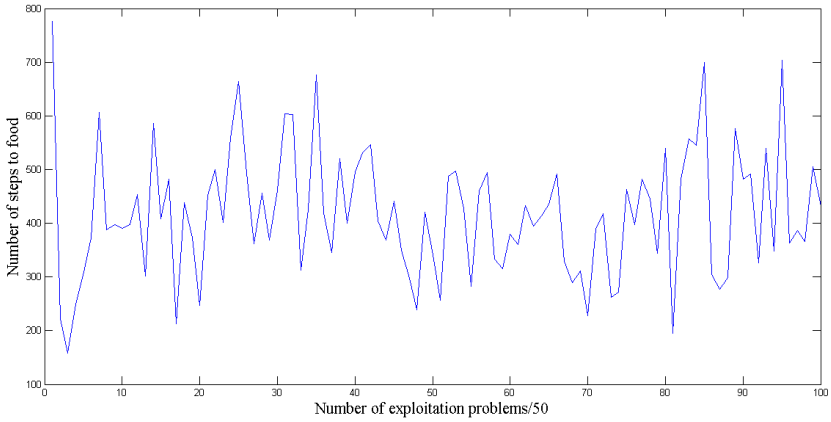


Figure 3-12: XCS animat in maze5 with subsumption.

The figures show the number of steps to food for animat at each problem. As it was discussed each problem is composed of a number of steps from a random blank point to reach to a food cell. The number of steps from a random blank cell to a food cell changes by learning of the animat through exploration and exploitation. At the first the number of steps is high because the animat doesn't know anything about the environment and its steps to food is nearly random. But after a number of problems it can reach to food faster and the number of steps to food decreases. These results are shown in this section. Animat after some problems acquires a population of classifiers that truly map the sensory information to actions and allow it to reach to a food cell. Subsumption mechanism is a way to make a smaller population of classifiers that contains general and useful ones.

At the first XCS algorithm without subsumption mechanism is considered. The effect of applying XCS algorithm for animat in woods1 environment leads to decrease the average number of steps to food to a value around 1.9 steps that is very close to the optimal performance. For woods2 also it is around 1.9 steps to food that decreases from a value near 27 (average number of random walk to reach to food in woods2). For maze5 that is a more complex environment because of the challenging distribution of foods and obstacles, the number of steps to food decreases to a value around 11 that is not near the optimal performance. So, we conclude that with using XCS algorithm without subsumption mechanism the animat can learn by creating an appropriate population of classifiers which directs it toward food by reaching close to the optimal performance or in some cases far from it.

The performance of learning of animat when the subsumption mechanism is turned on remains nearly the same for woods1 and woods2 and for both the number of steps to food reaches to a value close to average of 1.9 steps to food. For maze5, the number of steps to food in this case doesn't converge and changes between 150 steps to around 700 steps to food. It is because of the creation of over-general classifiers. So, for maze5 using subsumption mechanism is not recommended. Over-general classifiers are too general classifiers that their actions are right in some situations and wrong in other situations [2]. They have additional # to stay accurate. In some situations generality overcomes accuracy and the population of classifiers becomes full of over-general classifiers which can decrease the performance. It is because the GA cannot distinguish between an accurate classifier and an over-general classifier with the same payoff and reproduce it. For more theoretical work on over-general classifiers refer to [28].

3.4.2.4 Bad choice of parameters (simplest case)

The choice of parameters is based on the values presented in the literature. There are a lot of parameters but most of them are never changed and most articles use the same parameters. So, the parameters are hard-wired parameters, i.e. part of the architecture. In fact, it is the population and the number of classifiers that changes in different environment instead of parameters of the systems. As an example in multilayer artificial neural networks the values of the hyperparameters are set different in different problems to work in its best performance, but in XCS the parameters are nearly the same and instead the number of classifiers and the values of rules changes in different problems. The range of parameters setting is presented in [27]. However, if any parameter is chosen outside of appropriate range the XCS performance may be affected badly. For this reason woods1 is considered to show any bad choice of one parameter and its effect on the performance. $p_{\#}$ is set to 0.9 instead of 0.33. The result is presented in Figure 3-13.

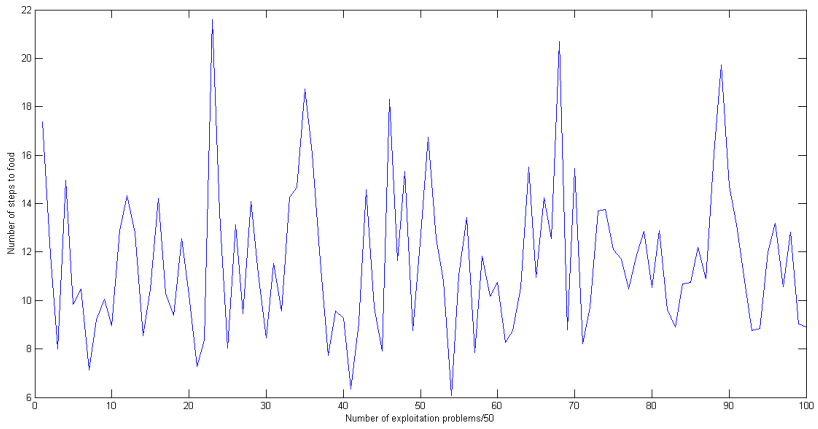


Figure 3-13: XCS animat in woods2 without subsumption for $p_{\#} = 0.9$

Experiments show that the learning of animat in woods1 is very sensitive to parameter $p_{\#}$ but it is not very sensitive to the other parameters as well. This conclusion can be different for different environments. For environments such as maze5 that over-general classifiers are produced, and is very sensitive to the subsumption procedure, change in θ_{sub} can have a high effect on the performance.

3.4.2.5 Why these performances occur?

It occurs because the animat learns to find food by exploring the environment to obtain payoff data at each state. The animat explores in odd problems and exploits in even problems. In other words the animat moves randomly in the environment to reach to a food. During this random search at each step it updates the performance, performance error, and fitness based on the achieved payoffs. However, the action is chosen randomly and not based on the maximum system prediction. In addition in the exploration problems the genetic algorithm is turned on and new classifiers are produced. So, in exploration problems the system is equipped with new rules and the weak classifiers are thrown away and the classifiers update their information about the payoff distribution in the environment.

The exploitation cycle is like test cycle in machine learning methods. The knowledge that the system has obtained is tested and the performance is measured. In exploitation the genetic algorithm is turned off because the population of classifiers should be kept fix to be tested for the performance that is the number of steps to food.

In nearly all the curves it is clear that the number of steps to food at the first is high, and it reaches to a low value after several iterations. This fact, tell that the system is learned for the corresponding environment and the population of classifiers map the payoff environment based on the mapping $X \times A \Rightarrow P$. The population of classifiers after much iteration can be used as a payoff classifier. The meaning of classifier systems is hidden in this task: the obtained population of classifiers classifies the environment based on the payoff. This kind of representation is a different representation than the usual classifiers that are used to classify data in a data mining task.

As it is visible in the environments, woods1 and woods2 are simple environments for animat to solve, because the number of sensory vectors is not high and reaching to food for animat is simpler than reaching to food in maze5. In woods1 and woods2 animat learns easily and the number of steps to food even in exploration problems is not very high. In maze5 that is a more difficult environment, the number of sensory vectors is higher, and there is only one food in the environment. When at the first step of each problem the place of animat is initialized in a random blank point of the environment, reaching to food in some situations can be a very long task especially in exploration problems that the actions are chosen randomly. So, a limit is considered for the number of steps to food in maze5 that doesn't allow the animat to try more than 1500 steps in a problem. The number of steps to food in maze5 is more than woods1 and woods2 because in average if a random blank point is chosen randomly in the environment as an initial point to start, reaching to food needs longer steps. So, the animat must learn a longer path to win a reward 1000. Using subsumption mechanism leads to producing over-general classifiers and when the number of over-general classifiers increases the performance decreases. So, **using subsumption mechanism in environments such as maze5 decreases the performance and another mechanism is needed to remove over-general classifiers.**

For the performance that is the number of steps to food, μ has an essential role because it determines the mutation operation in the genetic algorithm that has the main role to more general and less general classifiers. By running the algorithm, XCS shows sensitive to parameters χ and θ_{GA}. Especially if the value of θ_{GA} increases the number of times that the genetic algorithm executes decreases and the change in the number of classifiers decreases. It leads to decreasing the effect of generalization in the population. This doesn't have in general a good effect but in a case that over-general classifiers are produced in the population it is better to slow down the

effect of GA. The parameter ε_0 that is responsible for calculation of accuracy value has an important effect on producing classifiers that are more or less accurate.

3.4.3 Analysis of generalization in XCS

Why XCS doesn't converge to optimal solution when an over-general classifier appears in the population and what happens to the performance? What's the relation between environmental structure and the XCS performance?

An over-general classifier is a classifier that matches with different environmental niches where their rewards are different and thus they become inaccurate. In XCS, the fitness is based on accuracy and so, it tends to evolve general and accurate classifiers more. An over-general classifier can be deleted if it is inaccurate. A classifier to be inaccurate needs to be applied in distinct environmental niches. XCS may perceive an over-general classifier as accurate, because for XCS different rewards make a classifier inaccurate. However, this occurs if classifier happens in different environmental niches.

There are environments that animat doesn't visit all the situations of environment with the same frequency. In addition, there are situations that animat stays for a while and then takes another direction. In this situation over-general classifiers occur that are accounted as accurate. So, XCS instead of deleting them reproduce them and as XCS is based on accuracy it affects the performance of the system.

The animat doesn't converge to optimal policy if it doesn't see all the environmental niches frequently. In this way we observe that the exploration strategy is very important and should be chosen uniformly to explore the entire environment. In the original XCS in exploration cycle action selection is random. [24] has proposed a hypothesis related to the average random walk to food:

"*The smaller it is, the more likely the animat will be able to visit all positions in the environment frequently; The larger the average random walk is, the more likely the animat is to visit more frequently a certain area of the environment*."

When the niches of the environment are distant such as maze5, the animat can't change the niches as frequently as it is necessary to evolve an optimal policy. This is the reason that animat

works well for simple environments such as woods1 and woods2 and fails for more complex ones such as maze5.

If XCS doesn't explore all the niches of environment uniformly and the over-general classifiers that match to few niches of environment are very likely to be reproduced, then XCS fails to learn an optimal policy in the environment [29].

The reason that performance is poor for some environments is the problem in functioning of the generalization mechanism that leads to generation of over-general classifiers. In fact, the mechanism of XCS to delete over-general classifiers is very slow.

In some environments generalization capability prevents XCS from converging to optimal solution [30]. *Specify* is the name of an operator that slows down the generalization process and is a solution for the problem of creation of over-general classifiers to improve the performance of the system. This operator will be introduced in the next chapter.

3.5 A literature review on XCS animat approach

This section reviews different approaches and developments in XCS classifier system that is used as the main algorithm to deal with the Wilson's animat problem. The goal of the literature review on XCS animat here is to show that XCS algorithm is flexible enough for adding and changing many components and for creating variety of methods for different kinds of problems and situations.

Neuro and fuzzy XCS:

Neural XCS which is named X-NCS is presented in [31]. The idea is to provide a neural network (multi-layered) representation of the condition and action of XCS classifiers that GA is used to evolve the neural network. Fuzzy logic then is used through the radial basis function networks. The optimal performance of X-NCS is presented for single-step, multi-step, and function approximation tasks [32]. The use of back-propagation in conjunction with GA is then added in [33] and is tested for continuous and discrete action tasks. Building anticipations of the expected states by X-NCS is presented in [34]. Local search is a method used for difficult optimization problems that the algorithm moves among candidate solution by applying local changes to find an optimal solution. Combination of local search of back-propagation and global search of GA that creates a neural XCS is applied to X-NCS and is described in [35]. Fuzzy-XCS for single-

step reinforcement learning problems is presented in [36] and [37]. Using a fuzzy logic method to control the balance between exploration and exploitation rates of XCS is proposed in [38] and its extension is presented in [39]. A Spiking neural network (a network with dynamic internal states) representation of the condition and action of XCSF [40] classifiers that an evolutionary process is used to exploit parameter self-adaptation (the adaptation process to changes that have been occurred to change the condition to a new one) is presented in [41]. Constructivism is a theory that discusses about the structure of knowledge in human being and the interaction between existing knowledge and new information. Using self-adaptive constructivism in neural XCS and XCSF that leads to adaptive behavior of agent which is representational flexibility (the ability of making an appropriate representational choice) guided by environment is the subject of [42]. Using self-adaptive parameters and neural constuctivism in neural XCSF in which a feed-forward multi-layered perceptron network is used to represent the classifier conditions is presented in [43]. It is used to solve a continuous maze environment with continuous-valued actions, discrete-valued actions, and continuous-valued actions in continuous time and continuous space. A connectionist XCS that uses neural networks and classifier systems in combination and for controlling an autonomous agent is presented in [44].

XCSF:

After invention of XCS in 1995, XCSF was proposed by Wilson in 2001 and 2002 in [40] and [45]. In these papers the function approximation is learned using prediction estimation. Furthermore, weight vectors have been added to the classifiers which leads to piecewise linear approximation (a function approximation method with a function composed of straight lines). Three basic modifications of XCS to produce XCSF are: 1. changing binary string input to integer input, 2. considering a weight vector for classifiers to compute payoff prediction, and 3. modification in updating procedure of weights. Papers [46], [47], [48], [49], and [50] have applied XCSF for function approximation and single step-problems. The ‘Frog’ problem that has been introduced in [51] is used to illustrate three architectures for testing continuous action XCSF [52]. A new XCSF called XCSFCA is introduced in [53] to improve the performance of XCSF (that works with computed prediction for continuous payoff and numerical input) with computed continuous action that would be applicable for robotics which need numerical action. The continuous action classifier is desirable for applications such as robotics. Using XCSF for multi-step problems with continuous inputs is investigated in [54]. In this paper it is shown that XCSF

can evolve a compact population of accurate and maximally general classifiers and that population provides optimal solution to the problem. Using XCSF for reinforcement learning problems involving delayed rewards is presented in [55]. XCSF is used as a method for generalized reinforcement learning. By this method XCSF can evolve optimal and near optimal solutions for linear reinforcement learning problems. Application of XCSF animat problem in woods environments is presented in [56].

XCS-LP:

Classifier system for environments with continuous reinforcement is called XCS-LP and was introduced in [51]. Examples of continuous payoff environments are in control, robotics, and financial time-series. In this system the classifier's prediction is a continuous linear function of input x. The frog problem then was presented in this paper and was used to test XCS-LP. XCS-LP has two differences from XCS: inputs are real and a linear polynomial is used which determines prediction from x. Frog problem is a problem that the classifier system acts as a frog that senses a fly and learns to jump to the distance that the fly is located in it. The sensory signal is decreased with the distance between them monotonically and the range of action (jump) is in a continuous range.

SB-XCS:

Tim Kovacs in his PhD thesis introduced SB-XCS (strength based XCS) to compare XCS which is based on accuracy and traditional LCS that is based on strength [57]. The results of SB-XCS on 6-Multiplexer and Woods2 are presented in [58]. Two views of LCS are presented in [59]: Genetic Algorithm-based systems and Reinforcement Learning-based systems. It discusses that Genetic algorithm-based systems are better for XCS and Reinforcement Learning-based systems are better for SB-XCS.

The concepts of strong over general rules and fit over general rules have been introduced in [60]. This paper claims that strong over general rules are the main basis of SB-XCS. According to this paper, the strong over general rules depend on biases in the reward function that is introduced in the paper. Then design of fit over general rules for XCS is done by defining biases in the variance of the reward function.

Generalization in XCS:

A theory of generalization and learning in XCS was presented in [61]. It was started from the generalization hypothesis of XCS in which mentions that XCS algorithm produces accurate and maximally general classifiers and then presents a simple equation for generalization hypothesis. The analysis of generalization in XCSF and methods to improve its generalization capability are presented in [46]. Analysis of generalization capabilities of XCS in animat problem for grid-world environments have been presented in [62], [24], and [30]. In [30] the test is performed on Maze4 where XCS fails to reach to optimal performance and generalization capabilities prevent XCS to reach to the optimal solution. In [24] the test is performed on Maze6 and Woods14 and again it is shown that XCS fails to reach to optimal performance and generalization capabilities prevent XCS to reach to the optimal solution. A hypothesis then is presented to explain the results. In [62] the test is performed on Maze 5, Maze 6, and Woods14.

Application of XCS for robotic and Alife:

Extension of XCS named X-TCS for continuous environments for robotics without a priori discretization is presented in [63]. Using XCS for robot autonomous application is presented in [64]. It has presented two robotic tasks and tested XCS on them. These two tasks are reactive and non-reactive. The reactive task is a task that action depends only on the current sensory information. The non-reactive task is a task that involves some kind of memory to work in aliasing states. Using XCS with additional internal memory for a robotic task with a simulated Khepera in an aliasing environment and with noisy sensory data is tested for variety of problems [65]. A non-communicating predator/prey scenario using LCS is presented in [66]. A group of predators observe a prey collaboratively. Each predator is equipped with a single and independent XCS. In this paper a memory is considered for learners to store the history of the local actions and payoffs. Extending classifier systems to exchange information to improve the performance is developed in [67]. Two kinds of information are considered to be transferred: the information in signal pattern of collection of homogeneous classifiers and the information that is the result of given tasks to the agents to solve different parts of the original problem. The experiments are performed on 6-multiplexer and 11-multiplexer. Navigation of a robot with noisy sensors many times yields to perception aliasing problem that different situations in the environment are perceived identical for a robot. In [65] XCS is used with additional internal memory to overcome

this problem. The experiment is performed for four Woods-type problems on a Simulated Khepera.

Extension of XCS for Multistep and Maze and Woods problems:

Four modifications of XCS to improve performance in highly size-constrained populations have been presented in [63]. The tournament selection is applied to XCS in [68] and shows more parameter independent and more efficient in guidance of fitness exploiting. XCS with random and biased action-selection regimes is used in some multi-objective maze problems (a maze environment that the agent has more than one objective) in [69]. The rule linkage mechanisms are applied to XCS to solve non-Markov tasks [70]. The resulted XCS is called corporate XCS (CXCS). Lanzi defined stochastic environment as the environments that actions of agent are uncertain. In this type of environments he developed XCS for stochastic environments. Then an extension to XCS with a higher level of uncertainty was proposed that it can learn the optimal solution. This extension was named XCSμ [71]. It was then shown that XCSμ is a general version and it is the same as XCS when it is used for the deterministic environments. An extension for XCS that messy code is used instead of binary string condition is studied in [72].

XCSI:

The modification of XCS for integer inputs is presented in [73]. The new XCS is called XCSI. XCSI has additional modifications in mutation operator, covering, and subsumption. XCSI is applied for data mining applications.

XCSM:

XCSM was introduced by Lanzi in [74] and [75] and is XCS with addition of internal memory to be used for animat problem dealing with non-Markovian environments (partially observable environments) with aliasing states. Perceptual aliasing problem is a problem that two different situations in the environment perceive as the same (aliasing states). It is shown that XCSM can converge to optimal solution in simple environments but may fail to evolve an optimal solution in more complex ones. This paper has been tested on woods101, woods102, Maze7 that are non-Markovian environments. The analysis of XCSM to show why it fails to learn optimal solution in complex partially observable environments is described in [76]. It shows that memory management of XCSM doesn’t guarantee convergence to an optimal solution. An extension then is provided to XCSM and has been called XCSMH. XCSMH can learn optimal policy in all the

environments. In this paper the test environments are woods101, woods102, Maze7, and Maze10. In non-Markovian environments there are different cells that their sensory vectors are the same but two different actions should be performed to guide animat toward food (optimal action). The more advanced discussions about XCSM and XCSMH for more complex environments are discussed in [77]. To test for more complex environments, woods101$\frac{1}{2}$ is considered that includes four different states that animat perceives as the same but need four distinct optimal actions.

Gradient-Based XCS (XCSG):

The idea of updating the reward prediction using gradient descent method and its analysis on generalization is presented in [78], [26], and [79]. It shows more stable and reliable in multi-step environments.

Summary of important events in XCS

In summary the development of XCS and its further improvements were started at 1995 by first introduction of XCS by Wilson [1]. It then continued by introduction of XCSM and XCSMH for non-Markovian environments in 1998 and 2000 by Lanzi [74] and [77]. Then the idea of Integer-valued XCS (XCSI) were presented by Wilson in 2000-2001 to solve multiplexer problems [73]. SB-XCS in 2002 were introduced by Tim Kovacs as a strength-based XCS [57]. In 2002 Wilson introduced XCSF to approximate functions with a XCS-based method [45]. Larry Bull in 2002 presented X-NCS and X-NFCS as neuro- and neuro-fuzzy XCS algorithms to solve multiple problems such as function approximation [31].

3.6 Conclusion

In this chapter, XCS classifier systems were introduced and it was shown that it can solve different RL animat problems. Some well-known environments that the Wilson's animat problem can be applied for them were also introduced to be used in this project. The XCS animat that is a Wilson's animat problem with XCS algorithm as its control architecture was tested on woods1, woods2, and maze5 and results were presented. It was shown that XCS cannot solve maze5 environment because of production of over-general classifiers in the population of classifiers and this is because of the generalization ability of XCS. So, the generalization mechanism of XCS in

some situations should be slow down to overcome this problem by removing over-general classifiers. To this goal in the next chapter *Specify* operator will be introduced to deal with this problem. A gradient-based XCS also will be introduced as another method to improve performance of XCS by addition of gradient descent in the prediction updating mechanism that is a more general method than tabular Q-learning used in XCS to update prediction. At the end combination of XCS with *specify* operator and gradient-based XCS will be introduced.

CHAPITRE 4 DEVELOPMENTS IN XCS TO IMPROVE PERFORMANCE IN MARKOVIAN ENVIRONMENTS

4.1 Introduction to XCSS

XCS evolves accurate and maximally general classifiers with minimum population size for woods1 and woods2. However, in some environments only a few generalizations can be done. Actually, it fails to learn optimal solution in some situations and over-general classifiers are created. To deal with this problem Lanzi introduced a *Specify* operator [30] to make XCS adapt to Maze5, Maze6 and woods14 [30]. The generalization mechanism of XCS is studied in depth in [24] and a specific hypothesis is presented. The hypothesis says that XCS can't learn an optimal policy if it doesn't visit all the areas of the environment frequently. The *Specify* parameter is introduced for the situations that XCS can't converge to optimal solution.

4.1.1 *Specify* operator

Generalization in learning classifier systems are introduced by use of # in the condition of the classifiers. In those environments that the generalization leads to creating over-general classifiers and thus to poor performance, generalization should be slowed. Don't care symbols of # are used in three places: in initial population that alleles are set to # with probability $P_{\#}$, in covering that # are set randomly, in mutation that alleles are randomly changed. The first two are accounted as the initialization of the system and so, mutation is the main component of generalization. So, to slow down generalization in certain situations a mechanism should contrast the mutation.

Specify operator is introduced to help generalization mechanism of XCS in eliminating over-general classifiers from the population. *Specify* acts on the action set when there are significant number of over-general classifiers in the action set; and leads to replacement of over-general classifiers with more specific offspring. *Specify* uses prediction error ε_j to find classifiers that because of existence of some #s match with different conditions in the environment with different rewards (oscillating classifiers). *Specify* replaces don't care symbols in the classifiers with a certain criterion. The initialization of new classifiers that are generated by *Specify* is similar to initialization of offspring in GA.

The mechanism of *Specify* based on [30] is as follows:

'At each cycle the average prediction error in action set $[A]$ is denoted by $\varepsilon_{[A]}$ and the average prediction error in population set $[P]$ is denoted by $\varepsilon_{[P]}$. We also introduce parameter N_{S_p} of the *Specify* that set to a constant. If $\varepsilon_{[A]}$ is twice larger than $\varepsilon_{[P]}$ and the average number of updates of classifiers in $[A]$ is at least N_{S_p} times, then a classifier is selected randomly from $[A]$ with probability proportional to its prediction error. The selected classifier is used to create a new classifier (offspring) and the new one is inserted in the population and if it is necessary another is deleted. To create the new classifier (offspring) from the selected classifier, each # symbol in selected classifier is replaced with the corresponding digit of the input with probability P_{S_p}'.

The XCS algorithm with using of *Specify* mechanism is called XCSS. Using *specify* operator makes XCSS to learn in a greater number of environments. The diagram representing the operation of XCSS is illustrated in Figure 4-1.

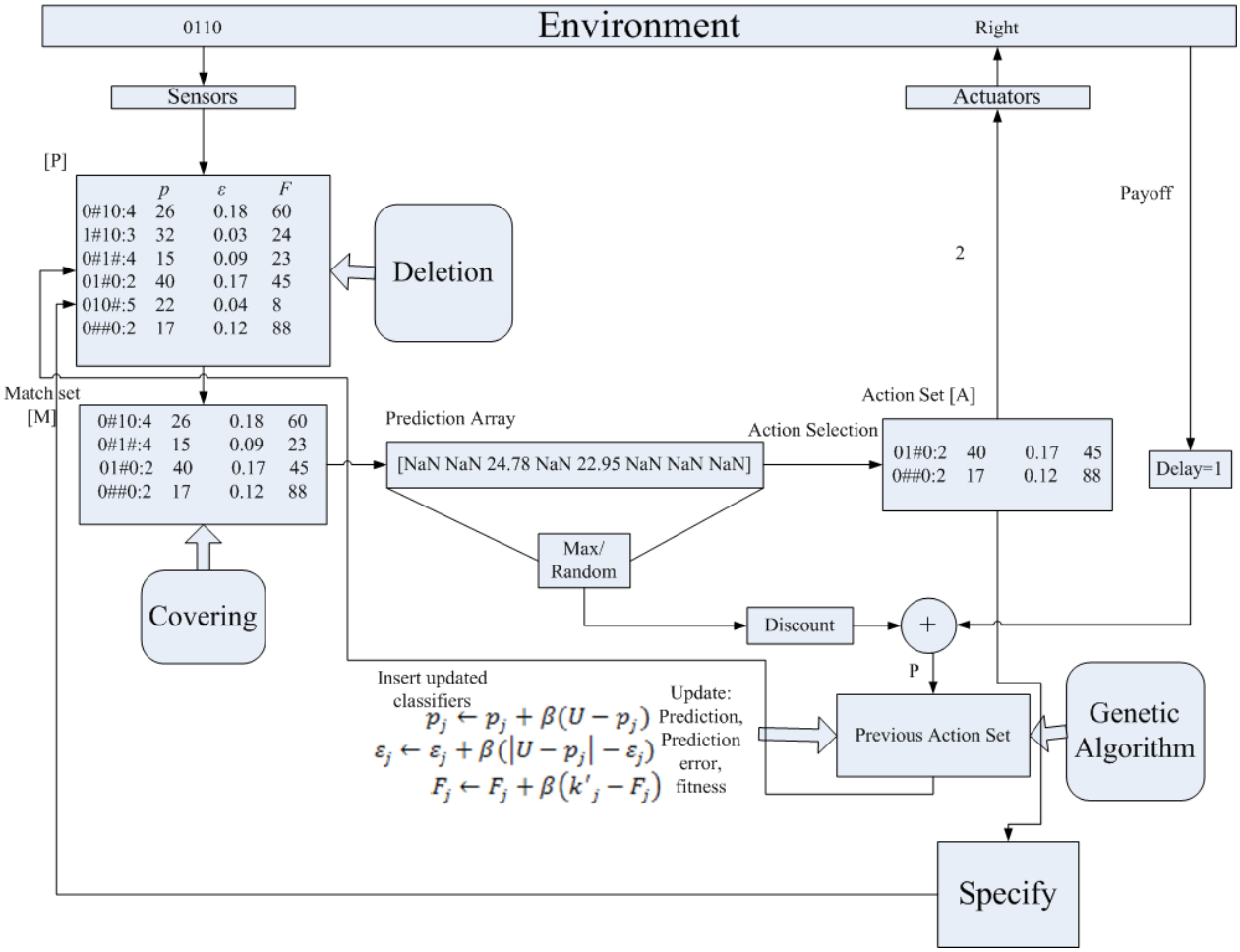


Figure 4-1: Block diagram of XCSS.

4.2 Using gradient descent in XCS to improve the performance in Markovian multi-step environments (XCSG)

As we discussed in the previous section XCS in some environments such as maze5 is not able to solve robustly. Using gradient descent in prediction updating mechanism in XCS is presented in this section to improve the performance of XCS. XCS is tightly linked to reinforcement learning and therefore, gradient-based methods in reinforcement learning that have been used for function approximation, are applicable to XCS. In multi-step environments that are modeled as a Markov Decision Process, Q-learning can be used to learn state-value function $Q(.,.)$ to predict the current reward. However, tabular Q-learning is infeasible for large problems. This is why function approximation methods based on gradient descent are used. XCSG in this part is presented based on [26].

4.2.1 Reinforcement learning and XCS

As we described in Chapter 2, in reinforcement learning problem an agent's goal is to maximize the long term cumulative reward that has achieved through interaction with the environment. In MDP environments with the finite state set S, finite action set A, at time t the agent senses the environment and perceives state s_t and based on this information the agent selects action a_t which changes its state to s_{t+1} and then based on the selected action and its state it receives the reward r_{t+1} in its next state. The goal of agent is to maximize expected payoff value:

$$E\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\right] \tag{4.1}$$

$0 \leq \gamma < 1$; The reinforcement learning methods are used to teach agent to maximize the expected payoff value by defining an action-value function $Q(.,.)$ which maps state-action pairs to the expected payoff value.

As we discussed before Q-learning is a well-known algorithms for solving reinforcement learning problems. $Q(s,a)$ is the predicted payoff when agent performs action a in state s. Q-learning iteratively approximates the optimal action-value function Q^* that maps state-action pairs to the expected reward. In fact, Q-learning approximates the table of $Q(s,a)$ values called Q-table.

Thus, this approximation is called tabular Q-learning. At the start for each state-action pair $Q(s,a)$ is initialized randomly at $t=0$. The agent at time $t-1$ in state s_{t-1} performs action a_{t-1} and at time t receives reward r_t and state s_t. At time t the $Q(s_{t-1},a_{t-1})$ is updated as:

$$Q(s_{t-1},a_{t-1}) \leftarrow Q(s_{t-1},a_{t-1}) + \beta\big(r + \gamma \max_{a\in A} Q(s_t,a) - Q(s_{t-1},a_{t-1})\big) \qquad (4.2)$$

Learning rate β is $0 \le \beta \le 1$. It certainly, converges to the optimal value of Q^* (for the proof of convergence see [20]).

Now, we have to investigate how XCS approximates Q^* values. As we discussed before, XCS is a RL method to solve RL problems. The generalization in XCS occurs because of the evolution of population of the classifiers that use # symbols. The population of classifiers in XCS represents the action-value function in RL. XCS contains a RL setting inside it and it makes population to approximate Q^*. In XCS the system prediction $P(a_{t-1})$ plays the role of $Q(s_{t-1},a_{t-1})$ in reinforcement learning and is represented by classifiers exist in the $[A]_{-1}$ and the system prediction $P(a_t)$ plays the role of $Q(s_t,a_t)$ and is represented by classifiers exist in the $[A]$:

$$Q(s_{t-1},a_{t-1}) = P(a_{t-1}) = \frac{\sum_{cl_j\in[A]_{-1}} p_j F_j}{\sum_{cl_j\in[A]_{-1}} F_j} \qquad (4.3)$$

The reason for this equality is because of the definition of state-action value that predicts the future payoff values. Using this definition appears in XCS update equation for prediction of a classifier:

$$p \leftarrow p + \beta(U - p) \qquad (4.4)$$

Where $U = r + \gamma \max_{a\in[A]} P(a)$. So, the prediction update for each classifier in the previous-time action set can be given by:

$$p \leftarrow p + \beta\left(r + \gamma \max_{a\in[A]_{-1}} P(a) - p\right) \qquad (4.5)$$

By comparison of updating procedure of XCS and Q-learning we see that the updating mechanism of XCS for prediction is inspired from tabular Q-learning.

4.2.2 XCS with gradient descent

Tabular Q-learning for large problems is infeasible because the table that maps state-action values to Q values grows exponentially by dimension and therefore, more experience is needed to converge to a good Q^* and more memory is needed to store the table. Generalization is a way to cope with complexity of environment to produce a good approximation of the optimal Q-table using a small memory by limited number of experiences. In reinforcement learning literature the generalization is made possible using online function approximation (to approximate Q^*) methods such as gradient descent techniques. So, gradient descent function approximation methods are used to approximate Q^*. Using gradient descent approximation methods are actually assigning a 3-D function that maps state-action pairs to payoff values. When the number of pairs increases the function tends to become similar to a surface. So, in function approximation a good estimation of such a payoff surface is developed [26].

In gradient descent approximation in Q-learning, the goal is to minimize the error between desired payoff value of the current state-action pair and the current payoff estimate over a certain approximator w that $Q(s_t, a_t)$'s are its functions:

$$\min_{\mathrm{w}} \beta\left(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})\right) \tag{4.6}$$

Depending on the definition of state-action values, weights w can be different. The change Δw for each weight w at each time step t is:

$$\Delta w = \beta\left(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})\right) \frac{\partial Q(s_{t-1}, a_{t-1})}{\partial \mathrm{w}} \tag{4.7}$$

The weights are updated based on the equation above. Function approximation methods that use this equation are called direct gradient descent algorithms.

Direct gradient descent algorithms are fast but sometimes unstable. So, residual gradient descent algorithms have been developed which are slower but more stable. The weight updates in Q-learning with residual gradient descent based on [26] are as follows:

$$\Delta w = \beta\left(r + \gamma \max_{a\in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})\right)\left(\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial \mathrm{w}} - \varphi\gamma \frac{\partial}{\partial \mathrm{w}}\left(\max_{a\in A} Q(s_t, a)\right)\right) \quad (4.8)$$

$\gamma \frac{\partial}{\partial \mathrm{w}} (\max_{a\in A} Q(s_t, a))$ estimates the effect of current modification of weight on the value of next state [26].

Based on the above explanations gradient descent can be added to the prediction update in XCS to improve learning capabilities of XCS. For each classifier cl_k in the action set (or previous time step action set) the gradient component is computed as follows:

$$\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w} = \frac{\partial}{\partial p_k}\left[\frac{\sum_{cl_j\in[A]_{-1}} p_j F_j}{\sum_{cl_j\in[A]_{-1}} F_j}\right] = \frac{F_k}{\sum_{cl_j\in[A]_{-1}} F_j} \quad (4.9)$$

So, first the sum of classifiers fitness in the action set is computed $F_{[A]_{-1}} = \sum_{cl_j\in[A]_{-1}} F_j$ and then the prediction of each classifier in $[A]_{-1}$ is updated as follows:

$$p_k \leftarrow p_k + \beta\left(r + \gamma \max_{a\in[A]_{-1}} P(a) - p_k\right)\frac{F_k}{F_{[A]_{-1}}} \quad (4.10)$$

The update of other parameters (prediction error and classifier fitness) remains without change. This type of XCS is named XCSG. The block diagram describing XCSG is illustrated in Figure 4-2.

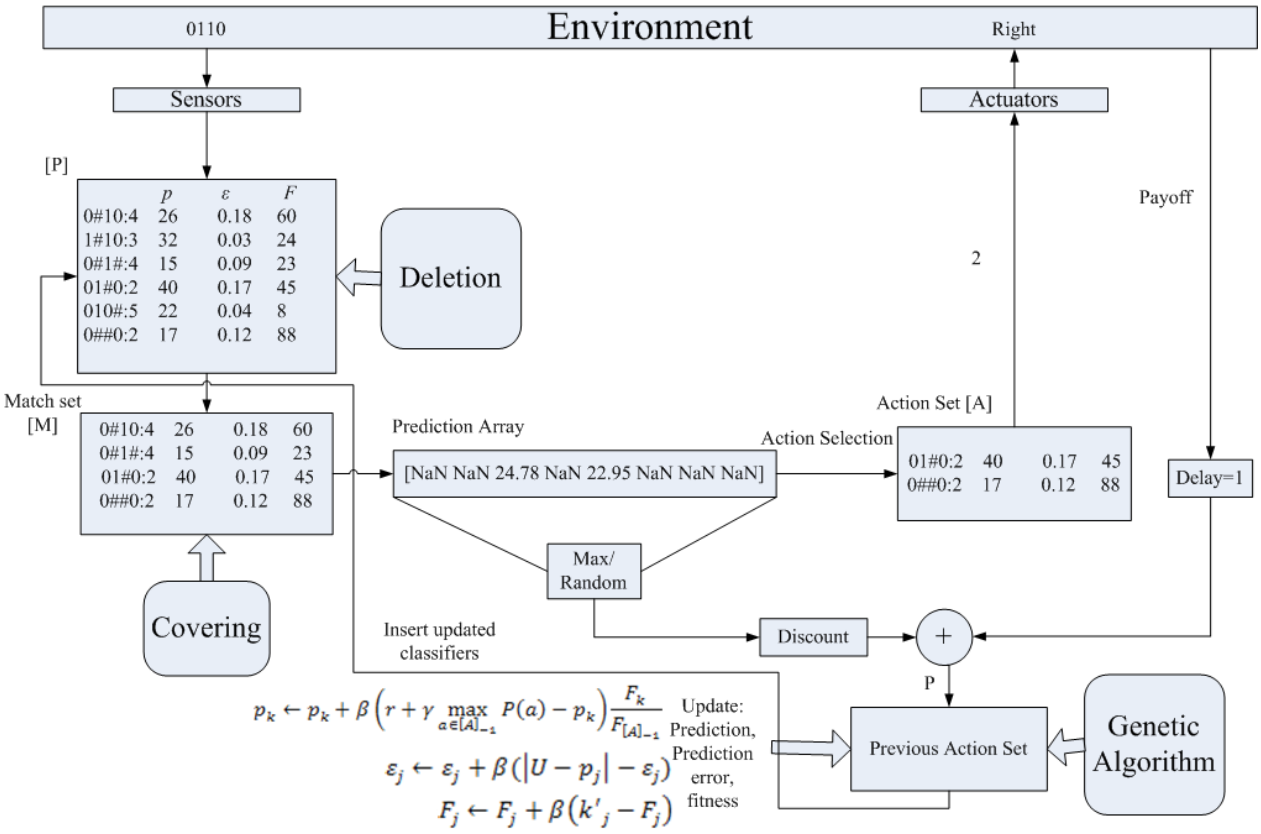


Figure 4-2: Block diagram of XCSG.

Term $\frac{F_k}{F_{[A]_{-1}}}$ adjusts the learning rate β adaptively for each classifier. For a classifier if $\frac{F_k}{F_{[A]_{-1}}}$ is small then the prediction update is based on small learning rate, and vice versa. It also has effect on accurate and over-general classifiers. For, over-general classifiers, $\frac{F_k}{F_{[A]_{-1}}}$ is small and the prediction is stable value. For, accurate classifiers (maximally general) the prediction converges to its actual value faster than inaccurate classifiers. So, inaccurate classifiers will have more reliable prediction by a small learning rate and accurate classifiers have a more reliable prediction because they are more accurate. Using gradient update the payoff surface would become more reliable and improve the generalization capability.

For residual gradient also weight update works as follows:

$$\Delta w = \beta\left(r + \gamma \max_{a\in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})\right)\left(\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial \mathrm{w}} - \varphi\gamma \frac{\partial}{\partial \mathrm{w}}\left(\max_{a\in A} Q(s_t, a)\right)\right) \quad (4.11)$$

To develop it for XCS $Q(s_t, a)$ is corresponding to the system prediction of action a in XCS. $\hat{a} = arg\max_{a\in A} P(a)$ is the action corresponding to the highest system prediction, the component $\frac{\partial}{\partial \mathrm{w}}(\max_{a\in A} Q(s_t, a))$ can be computed as

$$\frac{\partial}{\partial \mathrm{w}}\left(\max_{a\in A} Q(s_t, a)\right) = \frac{\partial}{\partial p_k}\left(\max_{a\in A} P(a)\right) = \frac{\partial}{\partial p_k} P(\hat{a}) = \frac{\partial}{\partial p_k}\left(\frac{\sum_{cl_j\in[M]|\hat{a}} p_j F_j}{\sum_{cl_j\in[M]|a} F_j}\right), \quad (4.12)$$

where $[M]|\hat{a}$ contains classifiers in $[M]$ that advocate action $\hat{a}$. To compute this value two cases should be considered:

1. If $cl_k \in [A]_{-1}$ also appears in $[M]|\hat{a}$ then $\frac{\partial}{\partial \mathrm{w}} \max_{a\in A} Q(s_t, a) = \frac{F_k}{\sum_{cl_j\in[M]|\hat{a}} F_j}$.
2. If $cl_k \in [A]_{-1}$ doesn't appear in $[M]|\hat{a}$ then $\frac{\partial}{\partial \mathrm{w}} \max_{a\in A} Q(s_t, a) = 0$.

So, at time step t for each $cl_k \in [A]_{-1}$:

$$\frac{\partial}{\partial \mathrm{w}}(\max_{a\in A} Q(s_t, a)) = res_k = \begin{cases} \frac{F_k}{\sum_{cl_j\in[M]|\hat{a}} F_j}, & if\ cl_k \in [M]|\hat{a}\,. \\ 0 & \text{otherwise.} \end{cases} \quad (4.13)$$

Thus, to update the prediction of classifiers it can be represented as

$$p_k \leftarrow p_k + \beta\left(r + \gamma \max_{a\in A} P(a) - p_k\right)\left[\frac{F_k}{F_{[A]_{-1}}} - \varphi\gamma res_k\right]. \quad (4.14)$$

The procedure to update the prediction is as follows:

First, the action $\hat{a}$ that is corresponding to the highest system prediction should be computed, then a set $[\hat{A}]$ is created containing all classifiers of $[M]$ with action $\hat{a}$. Then the parameters of classifiers of $[A]_{-1}$ are updated using the sum $\hat{F}$ of classifier fitnesses in $[A]_{-1}$

$$F_{[A]_{-1}} = \sum_{cl_k \in [A]_{-1}} F_j \ , \tag{4.15}$$

$$F_{[\hat{A}]} = \sum_{cl_k \in [\hat{A}]} F_j \quad . \tag{4.16}$$

At the end the prediction of each classifier in $[A]_{-1}$ that also exists in $[\hat{A}]$ is updated:

$$p_k \leftarrow p_k + \beta \left(r + \gamma \max_{a \in A} P(a) - p_k \right) \left[\frac{F_k}{F_{[A]_{-1}}} - \gamma \frac{F_k}{F_{[\hat{A}]}} \right] . \tag{4.17}$$

Otherwise, if classifier is not in $[\hat{A}]$, the classifier will be updated based on the gradient approach:

$$p_k \leftarrow p_k + \beta \left(r + \gamma \max_{a \in A} P(a) - p_k \right) \left[\frac{F_k}{F_{[A]_{-1}}} \right] . \tag{4.18}$$

4.3 XCSSG : combination of using *Specify* operator in gradient-based XCS

As it was described in Chapter 3, XCS has problem in environments such as maze5 because of creation of over-general classifiers. To solve this problem *Specify* operator [30] was presented to remove over-general classifiers from the population. The gradient-based XCS [26] was also introduced to improve the performance of XCS in complex environments by improving the prediction adjustment mechanism of XCS. So, two improvements are considered to improve the performance of XCS, one is improvement in discovery mechanism and the other is in reinforcement mechanism. In other words the first improves the evolutionary mechanism of animat's control architecture and the second one improves learning mechanism.

XCSSG is a new algorithm that integrates these two mechanisms at the same time. It applies *Specify* operator and gradient-based mechanism at the same time to improve both the evolution and learning of the Wilson's animat in various environments. So, in XCS the *Specify* operator works as described in 4.1.1 and the prediction mechanism applies prediction update of equation 4.10 instead of 4.5. The block diagram for description of XCSSG is illustrated in Figure 4-3.

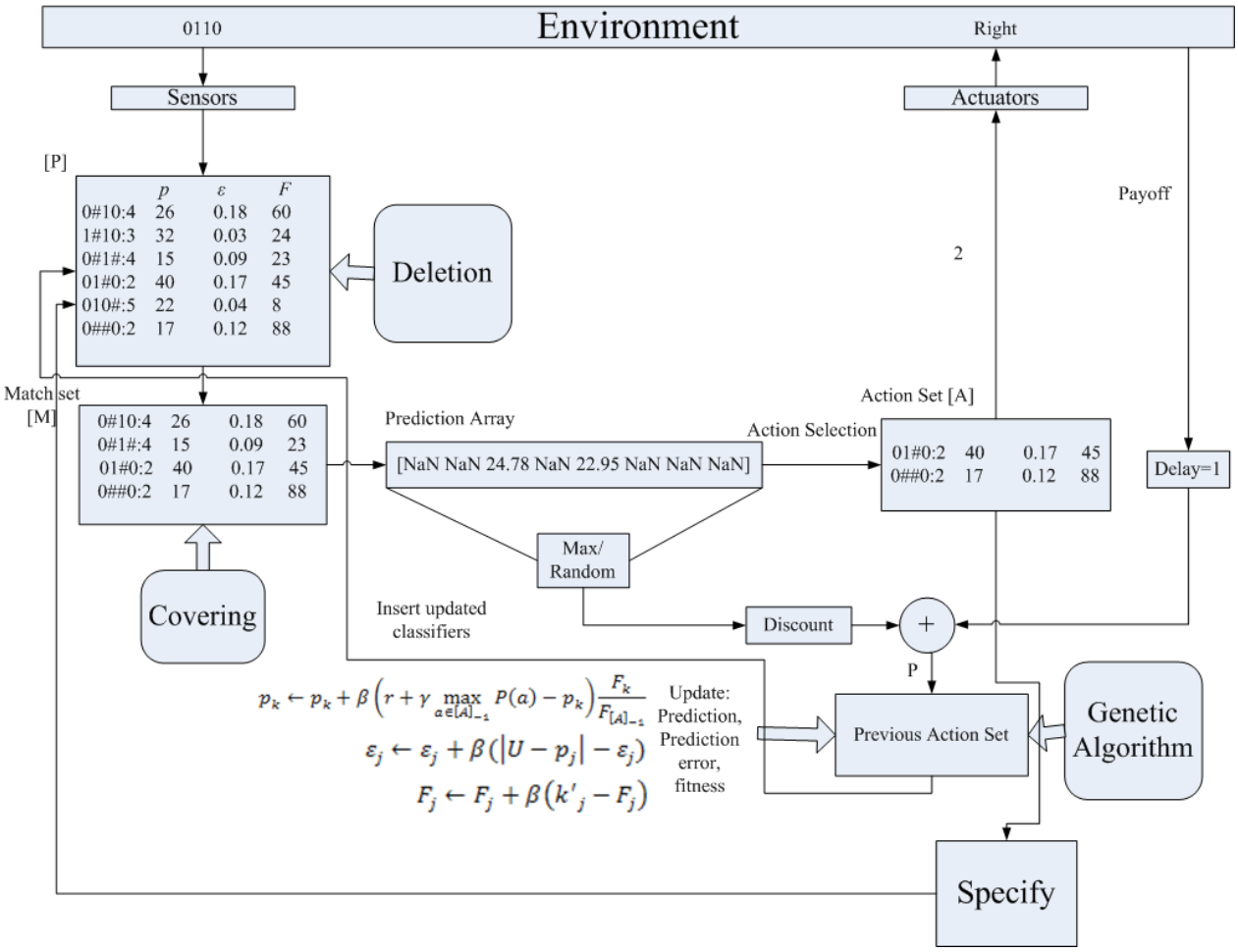


Figure 4-3: Block diagram of XCSSG.

4.4 Results for XCSS, XCSG, and XCSSG in various two-dimensional environments and their comparison

In this section the developed XCS animat problems have been tested in woods1, woods2, and maze5. The developed XCS algorithms that have been tested to result improvement in the performance are XCSS and XCSG. Two versions of gradient-based XCSG i.e. XCS with gradient descent and with residual gradient descent have been implemented and tested. At the end XCSG and XCSS are combined and resulted in a new XCS that we have called XCSSG in which both *Specify* and gradient mechanisms are applied to the corresponding animat problem in various environments. Performance is computed as the average number of steps to food for the last 50

exploitation cycles. If the food is not still achieved after execution of 1500 steps in one problem, the next problem starts.

4.4.1 XCSS

The list of parameters for experiment of animat problem with XCSS in each environment is presented in Table 4.1.

Table 4.1: List of parameters for experiment of animat problem with XCSS in each environment

	N	p_I	ε_I	F_I	β	γ	ε_0	α	ν	θ_{GA}	χ	μ	$P_\#$	θ_{del}	δ	θ_{mna}	θ_{sub}	p_{explr}	N_{sp}	P_{sp}
Woods1	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1	20	0.5
Woods2	800	10	0.00001	10	0.2	0.71	10	0.1	5	25	0.8	0.01	0.33	20	0.1	8	20	1	20	0.5
Maze5	2500	10	0.00001	10	0.2	0.71	5	0.1	5	25	0.8	0.01	0.3	20	0.1	8	20	1	20	0.5

Results of the XCSS algorithm in woods1, woods2, and maze5 are presented in Figure 4-4 to Figure 4-6.

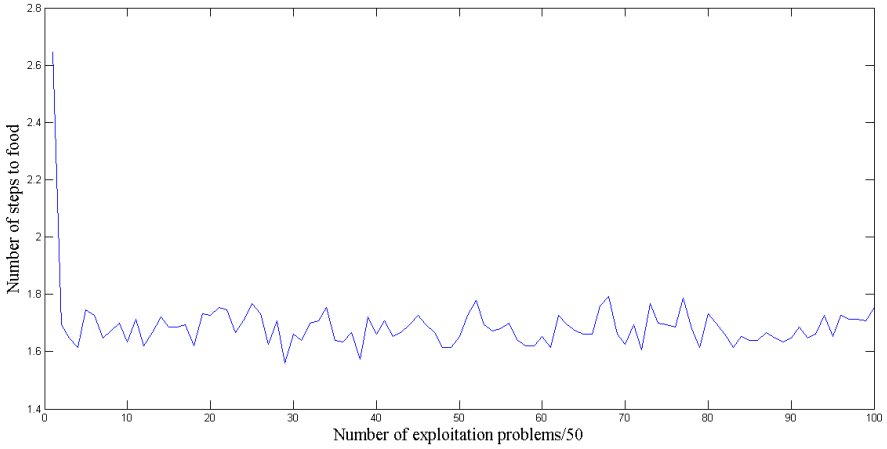


Figure 4-4: XCSS animat in woods1.

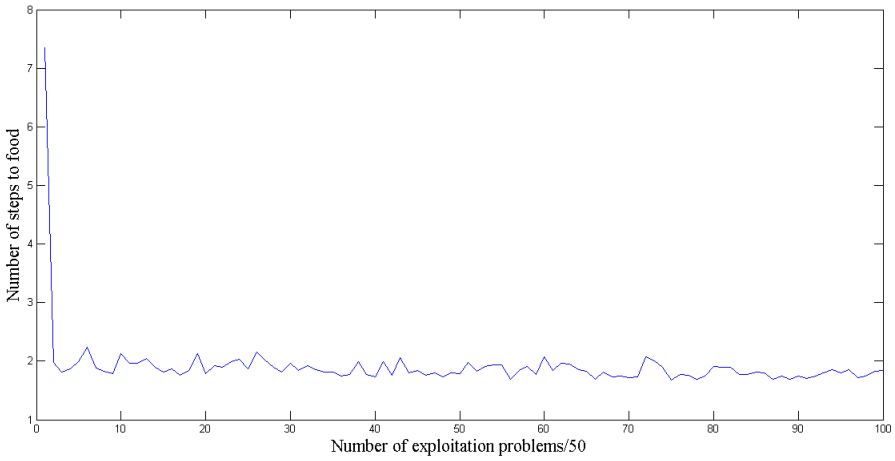


Figure 4-5: XCSS animat in woods2.

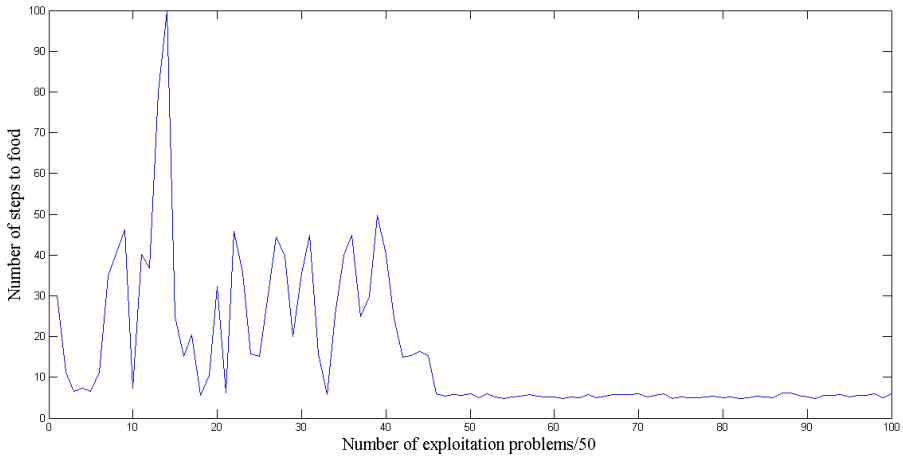


Figure 4-6: XCSS animat in maze5.

The performance of XCSS in woods1 and woods2 is stable and fast and approaches to the optimal performance around 1.7 in woods1 and 1.9 in woods2. In maze5 XCSS reaches to the optimal performance that is around 5. In maze5 the performance is slow because the number of steps to food converges to the optimal performance after approximately 4500 problems.

4.4.2 XCSG direct

The list of parameters for experiment of animat problem with direct XCSG in each environment is presented in Table 4.2.

Table 4.2: List of parameters for experiment of animat problem direct XCSG in each environment

	N	p_I	ε_I	F_I	β	γ	ε_0	α	ν	θ_{GA}	χ	μ	$P_\#$	θ_{del}	δ	θ_{mna}	θ_{sub}	p_{explr}
Woods 1	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Woods 2	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Maze5	3000	0.0001	0.00001	0.001	0.2	0.71	5	0.1	5	30	0.8	0.02	0.2	20	0.1	8	20	1

Results of the direct XCSG algorithm in woods1, woods2, and maze5 are presented in Figure 4-7 to Figure 4-9.

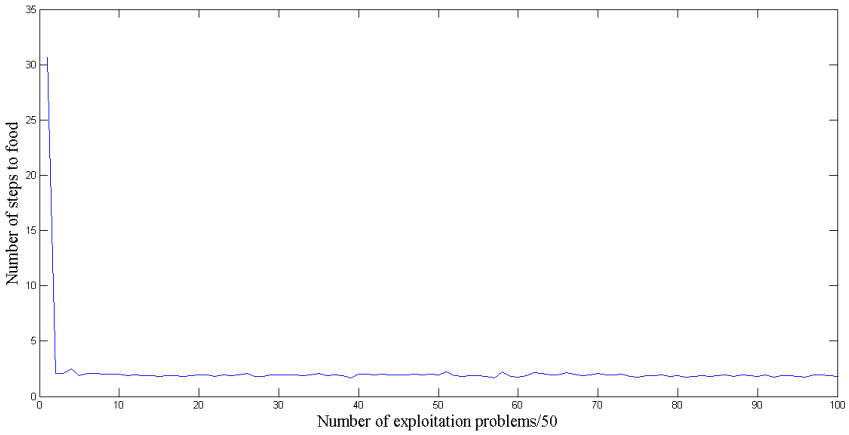


Figure 4-7: XCSG animat in woods1.

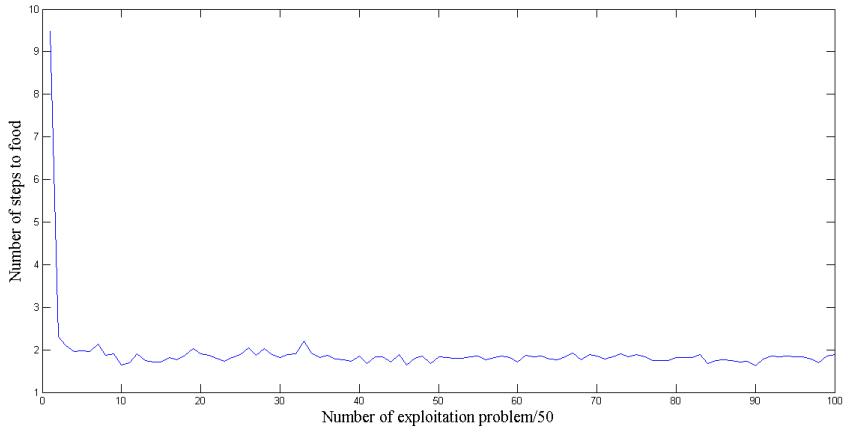


Figure 4-8: XCSG animat in woods2.

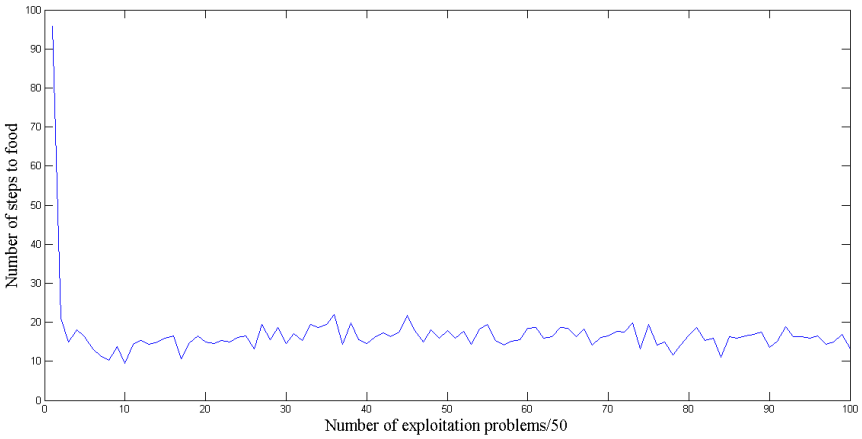


Figure 4-9: XCSG animat in maze5.

The performance of direct XCSG in woods1 and woods2 approaches to value around 1.9. In maze5, the performance reaches to value around 15 that is bigger than what XCSS approaches. The speed is faster than XCSS.

4.4.3 XCSG residual

The list of parameters for experiment of animat problem with residual XCSG in each environment is presented in Table 4.3.

Table 4.3: List of parameters for experiment of animat problem with residual XCSG in each environment

	N	p_I	ε_I	F_I	β	γ	ε_0	α	ν	θ_{GA}	χ	μ	$P_\#$	θ_{del}	δ	θ_{mna}	θ_{sub}	p_{explr}
Woods1	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Woods2	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1
Maze5	3000	0.0001	0.00001	0.001	0.2	0.71	5	0.1	5	30	0.8	0.02	0.2	20	0.1	8	20	1

Results of the residual XCSG algorithm in woods1, woods2, and maze5 are presented in Figure 4-10 to Figure 4-12.

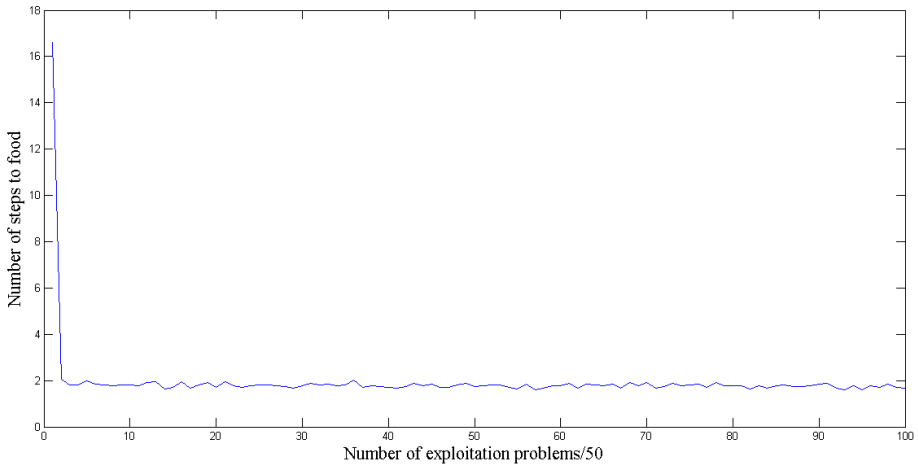


Figure 4-10: Residual XCSG animat in woods1.

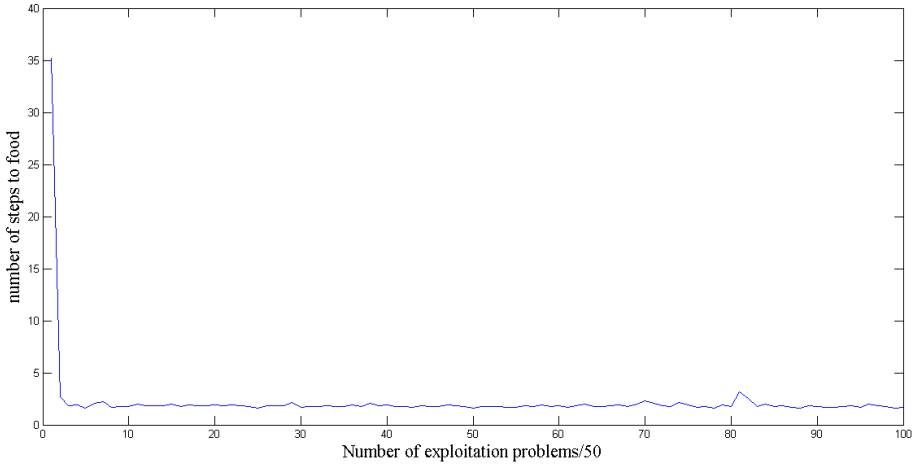


Figure 4-11: Residual XCSG animat in woods2.

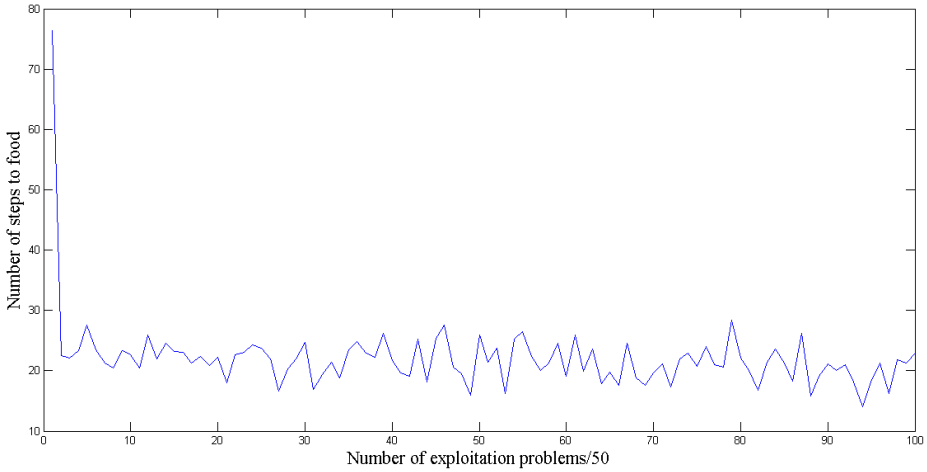


Figure 4-12: Residual XCSG animat in maze5.

Residual XCSG in woods1 and woods2 reaches to value around 1.9 which shows that it works well in these two environments. However, in maze5 its performance reaches to a value around 20 that in comparison works weaker than XCSS but is faster.

4.4.4 XCSSG

The list of parameters for experiment of animat problem with residual XCSSG in each environment is presented in Table 4.4.

Table 4.4: List of parameters for experiment of animat problem with residual XCSSG in each environment.

	N	p_I	ε_I	F_I	β	γ	ε_0	α	ν	θ_{GA}	χ	μ	$P_\#$	θ_{del}	δ	θ_{mna}	θ_{sub}	p_{explr}	N_{sp}	P_{sp}
Woods1	800	0.0001	0.00001	0.001	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1	20	0.5
Woods2	800	0.0001	0.00001	0.01	0.2	0.71	10	0.1	5	25	0.8	0.04	0.33	20	0.1	8	20	1	20	0.5
Maze5	2750	10	0.00001	10	0.2	0.71	5	0.1	5	28	0.8	0.01 5	0.25	20	0.1	8	20	1	20	0.5

Results of the direct XCSG algorithm in woods1, woods2, and maze5 are presented in Figure 4-13 to Figure 4-15.

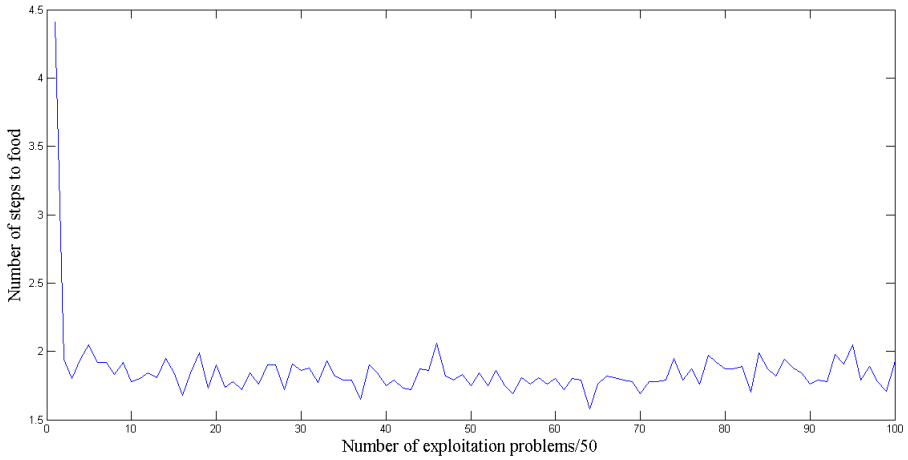


Figure 4-13: XCSSG animat in woods1.

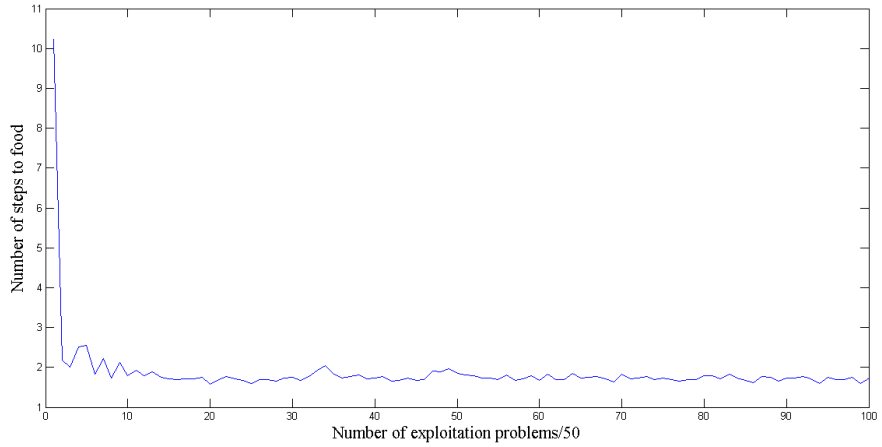


Figure 4-14: XCSSG animat in woods2.

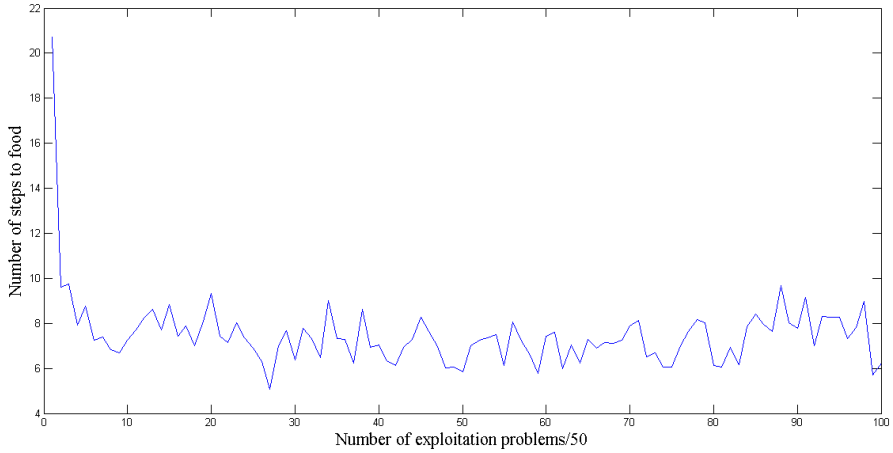


Figure 4-15: XCSSG animat in maze5.

XCSSG as a combination of XCSS and XCSG in woods1 and woods2 approaches to the optimal performance to a value around 1.9. Its performance in maze5 approaches to the average of around 7 steps to food that is close to what XCSS reaches but very faster. So, the speed of XCSSG is faster in comparison to XCSS and approaches near to the optimal value.

4.5 Comparison of results

To compare the results of different algorithms in each environment the performance curves are plotted in one figure to provide a criterion for comparison. The comparison results for each environment are illustrated in Figure 4-16 to Figure 4-24.

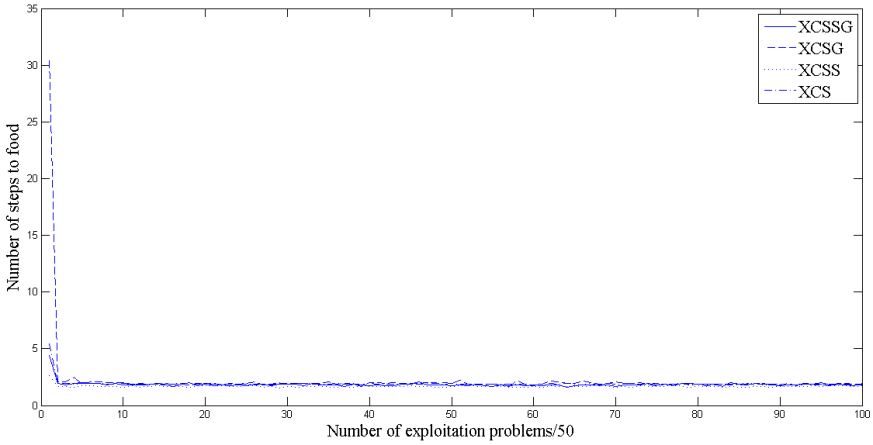


Figure 4-16: Comparison of XCS without subsumption, XCSS, direct XCSG, and XCSSG in woods1.

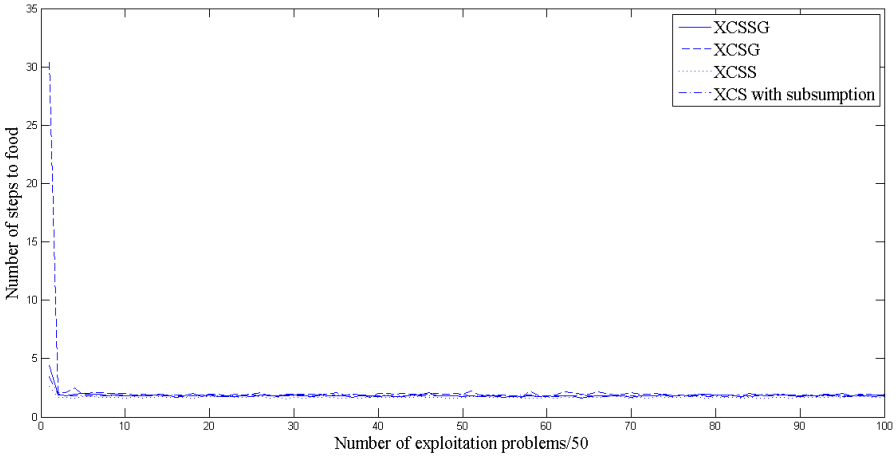


Figure 4-17: Comparison of XCS with subsumption, XCSS, direct XCSG, and XCSSG in woods1.

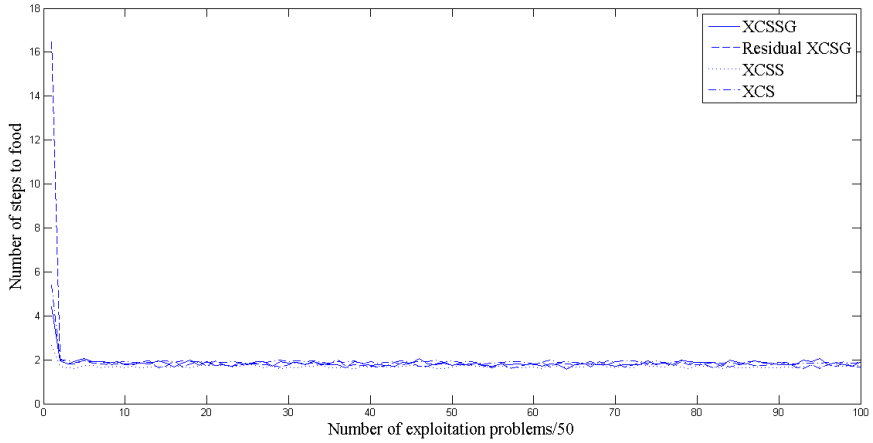


Figure 4-18: Comparison of XCS without subsumption, XCSS, Residual XCSG, and XCSSG in woods1.

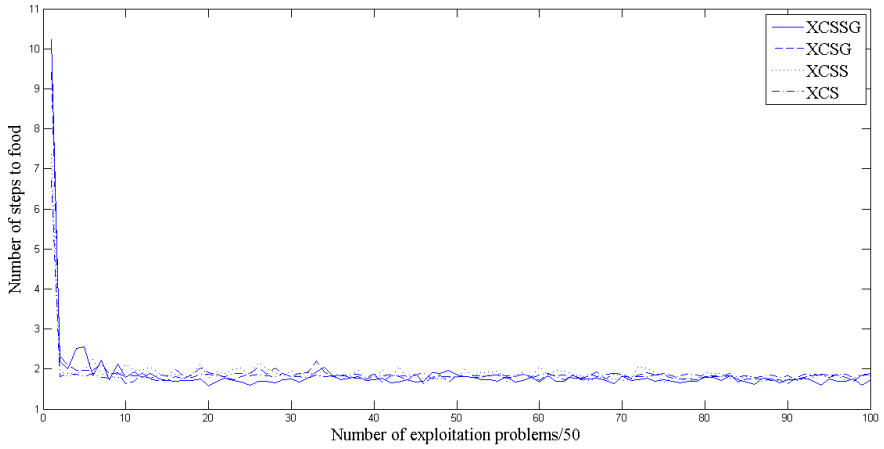


Figure 4-19: Comparison of XCS without subsumption, XCSS, direct XCSG, and XCSSG in woods2.

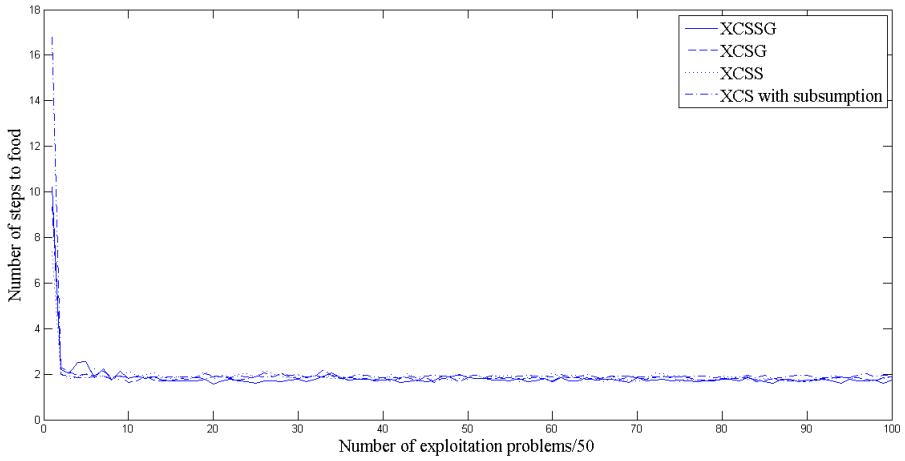


Figure 4-20: Comparison of XCS with subsumption, XCSS, direct XCSG, and XCSSG in woods2.

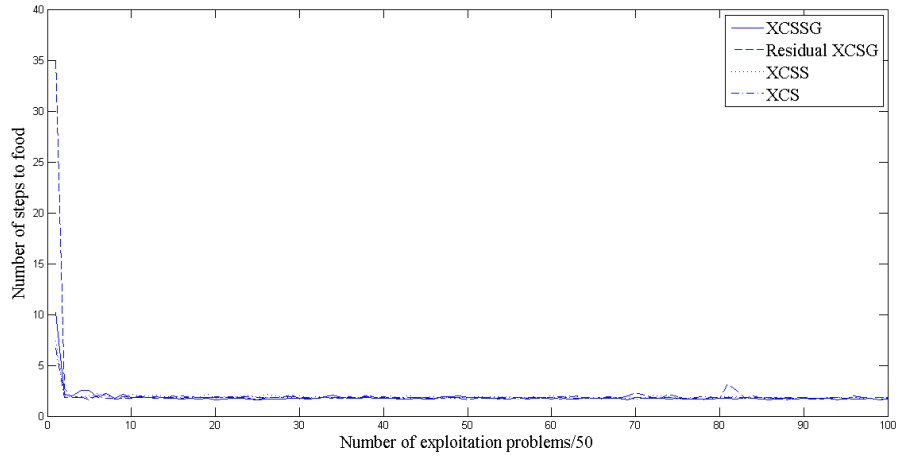


Figure 4-21: Comparison of XCS without subsumption, XCSS, Residual XCSG, and XCSSG in woods2.

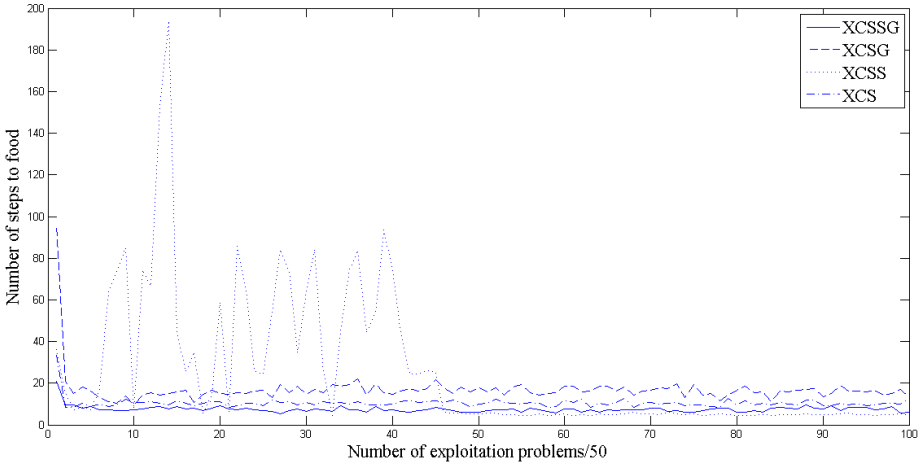


Figure 4-22: Comparison of XCS without subsumption, XCSS, direct XCSG, and XCSSG in maze5.

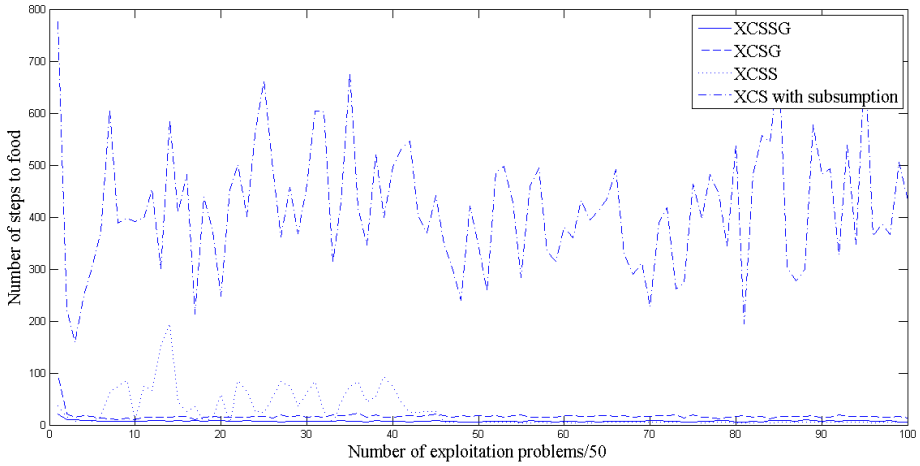


Figure 4-23: Comparison of XCS with subsumption, XCSS, direct XCSG, and XCSSG in maze5.

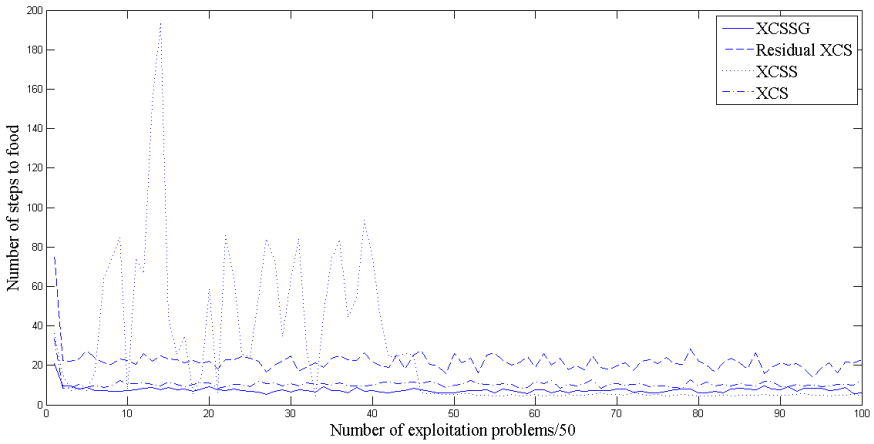


Figure 4-24: Comparison of XCS without subsumption, XCSS, Residual XCSG, and XCSSG in maze5.

Results show that XCSSG converges very close to the optimal performance fast and stably in maze5. XCSS converges to the optimal performance but very slow, and XCS and XCSG converge to a value different from optimal performance but not very far. XCS with subsumption doesn't converge even to a value close to the optimal performance. It is because of generation of over-general classifiers in the population set and the subsumption mechanism that removes classifiers without any look to the classifier if it is over-general or not.

The performance of different algorithms in woods1 and woods2 are very close.

Comparison with the Q-learning

To compare the obtained results with a more well-known reinforcement learning method that is better known in artificial intelligence context, Q-learning is chosen to be used as a method dealing with the animat problem in woods1 and maze5. So, the results of applying XCS are compared with the results obtained by using Q-learning.

In Q-learning a Q-table is assumed that contains $Q(s_t, a_t)$ values for each couple of state and action. The Q values update each time that agent situated in the corresponding state and performs that corresponding action. In other words, when agent is in state s_t and has received reward r_t for the action a_{t-1} in state s_{t_1}, the value $Q(s_{t-1}, a_{t-1})$ is updated as represented in Equation 4.2. At each step agent tries to choose that action with the highest $Q(s_t, a_t)$ value. Using this method, the number of actions to food decreases that shows the animat has learned to reach the food and the

values of the Q-table are stable. The results of using Q-learning for woods1 and maze5 are represented in Figure 4-26 and Figure 4-28.

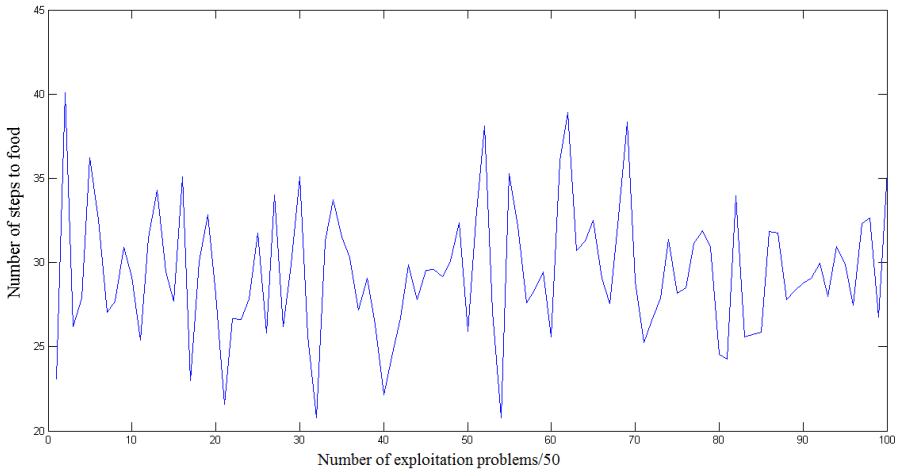


Figure 4-25: Random moves of animat in woods1 toward a food.

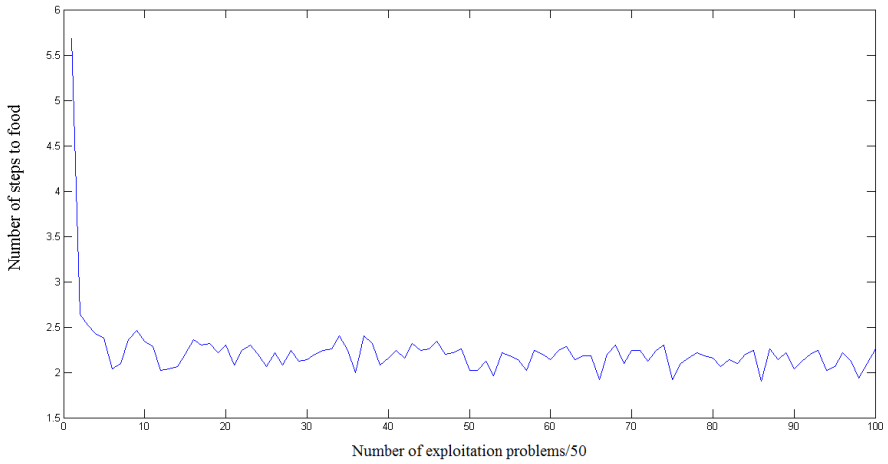


Figure 4-26: Applying Q-learning algorithm to solve the animat problem in woods1.

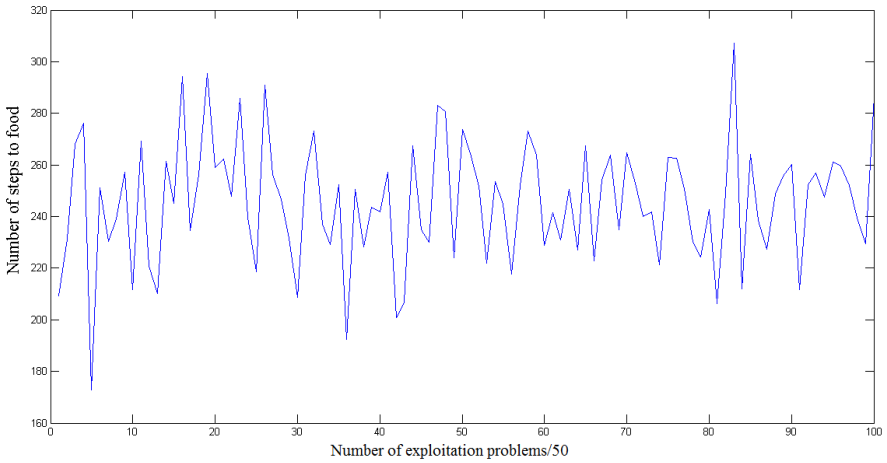


Figure 4-27: Random moves of animat in maze5 toward a food.

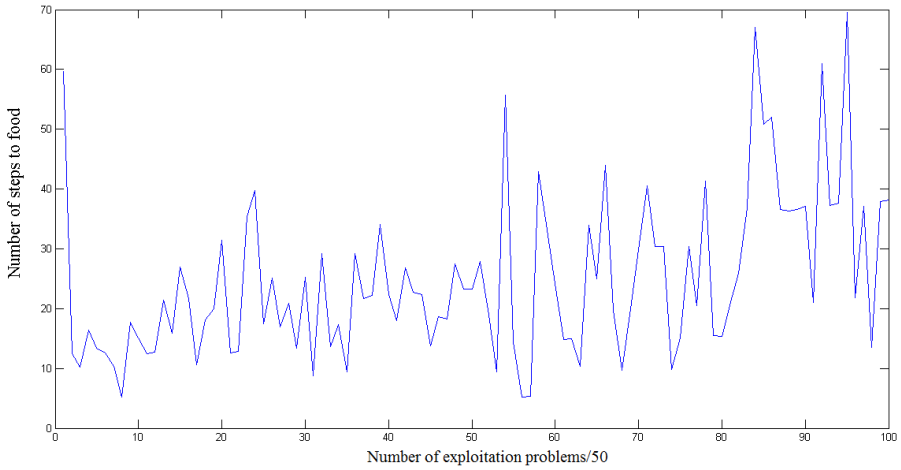


Figure 4-28: Applying Q-learning algorithm to solve the animat problem in maze5.

The results of using Q-learning in woods1 and maze5 show that the performance of XCS is higher than Q-learning. Figure 4-25 and Figure 4-27 represent the number of steps to food in woods1 and maze5 that are useful to compare when an adaptive learning algorithm is used to learn to find the food and when no algorithm is used and actions are random. The results show that the number of random steps to food in woods1 is around 30 steps and in maze5 are around 240 steps. When Q-learning is used the number of steps to food for woods1 decreases to around

2.2 and in maze5 it decreases to around 20. Using XCSSG leads to average of around 1.8 steps to food in woods1 and average of around 7 steps to food in maze5.

4.6 Conclusion

In this chapter two main developments for XCS were introduced to improve the performance of XCS for more complex problems. In XCS generalization mechanism works on environmental niches. *Specify* operator recovers dangerous situations in these niches. The convergence of the above environments all depends on the generalization capability of the system. It can be concluded that generalization in maze5 needs larger number of classifiers to completely learn the environment and large population needs more time and larger number of problems before converging to a small set of maximally general classifiers. In maze5 XCSS converges to the optimal value but very slow. XCSG that is a gradient-based XCS results a performance no better than XCSS but faster. Performance of XCSSG that is a combination of XCSS and XCSG is fast and converges to the optimal performance. In chapter5 new environments and new animat scenarios are introduced to give insight into the ability of XCS-family algorithms in learning at different situations and for different problems beyond the traditional works on XCS animat.

CHAPITRE 5 BEYOND THE TRADITIONAL XCS ANIMAT

5.1 Introduction

In the previous chapters the XCS animat problem was studied in detail and it was shown that the XCS-family (XCS, XCS with subsumption, XCSS, XCSG, and XCSSG) animat can learn in woods1, woods2, and maze5. In the literature on learning of the XCS animat in Markovian environments also, many papers use these environments. In addition, the animat has only the ability to sense its one step surrounding environment and decide based on that. Therefore, the ability of XCS animat cannot be shown for higher range of Markovian environments. In this chapter several new maze environments with different size and distribution of objects are introduced to test the learning ability of XCS animat in finding food.

The ability of XCS animat in changing environment gives a deeper insight into the adaptation ability of XCS algorithms. To experiment this ability, in this thesis a simple unstable resource problem is designed and different XCS-family algorithms are tested to present the adaptation ability of animat in an environment with a moving food. Competition between two XCS-family algorithms can give us better insight about the comparison of two algorithms. For this purpose a platform for competition of two XCS-family algorithms based on competitive Lotka-Volterra equation is designed and is tested. In addition to the previous experiments on Markovian environments, for an animat with higher vision ability a non-Markovian environment can be a Markovian environment because at each time it obtains more information. To test the learning ability of XCS (and XCSSG) animat in several non-Markovian environments that are Markovian when animat has higher vision ability, several non-Markovian environments are designed and the performances are compared.

5.2 Environment generator and S2DM environments

To show the learning ability of XCS in general, we should design new environments randomly, check if they are Markovian or non-Markovian, and test the ability of XCS algorithm in learning these new environments. To make our system automatic and provide a platform for the future

research, we have built an environment generator that creates random maze environments with the size of interest and checks out if the environment is Markovian or non-Markovian. The objects in the environments that are generated by this environment generator are food and obstacles. The generator checks the environment if there is some states with the same sensory information to predict how well XCS-family algorithms can learn in a random environment. By using this tool we have designed five maze environments 5MS2DM2 (5 by 5 Markovian square 2-dimensional maze with 2 obstacles), 6MS2DM3 (6 by 6 Markovian square 2-dimensional maze with 3 obstacles), 7MS2DM6 (7 by 7 Markovian square 2 dimensional maze with 6 obstacles), 7nMS2DM6 (7 by 7 non-Markovian square 2 dimensional maze with 6 obstacles), and 7MS2DM8 (7 by 7 Markovian square 2 dimensional maze with 8 obstacles) environments.

5MS2DM2 is a small 5×5 Markovian environment containing food and obstacles (Figure 5-1). The optimal performance that is the average of minimum number of steps to food is calculated as the average of minimum steps that animat starts from any random blank point in the environment and reaches to food. For 5MS2DM2 environment the optimal performance is calculated as: $\frac{2+1+2+1+2+2}{6} = 1.66$.

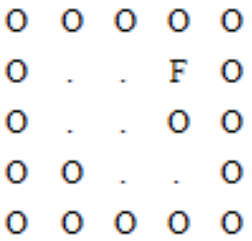


Figure 5-1: 5MS2DM2 environment.

6MS2DM3 is a 6×6 Markovian environment containing food and obstacles (Figure 5-2). For 6MS2DM3 environment the optimal performance is calculated as: $\frac{3+2++2+2+1+1+1+2+1+1+1}{12} = 1.58$.

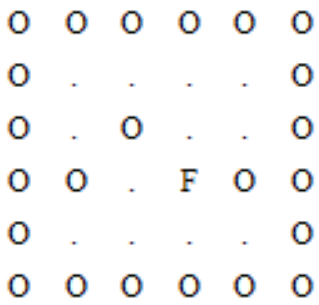


Figure 5-2: 6MS2DM3 environment

7MS2DM6 is a 7×7 Markovian environment containing food and obstacles (Figure 5-3). For 7MS2DM6 environment the optimal performance is calculated as:

$\frac{3+2+1+1+3+1+3+2+1+1+1+2+2+2+3+3+3+3}{18} = 2.05.$

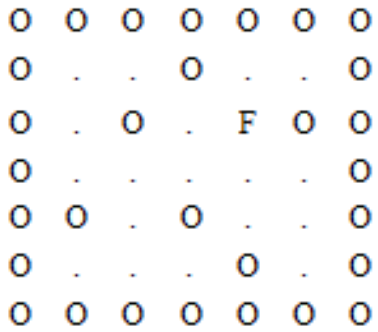


Figure 5-3: 7MS2DM6 environment

7nMS2DM6 is a 7×7 simple non-Markovian environment containing food and obstacles (Figure 5-4). In 7nMS2DM6 there are two positions with the same sensory string but their optimal action can be different. The positions 8 and 13 have the same sensory string, and also, positions 5 and 11 have the same sensory string too, see Figure 5-5. This similarity in the sensory information shows that the environment is non-Markovian. For 7nMS2DM6 environment the optimal performance is calculated as: $\frac{3+2+1+1+3+1+3+2+1+1+3+2+2+2+4+3+3+3}{18} = 2.22.$

```
O O O O O O O
O . . O . . O
O . O . F O O
O . . . O . O
O . O . . . O
O . . . O . O
O O O O O O O
```

Figure 5-4: 7nMS2DM6 environment

```
O  O  O  O  O  O  O
O  1  2  O  3  4  O
O  5  O  6  F  O  O
O  7  8  9  O  10 O
O  11 O  12 13 14 O
O  15 16 17 O  18 O
O  O  O  O  O  O  O
```

Figure 5-5: numbered 7nMS2DM6 environment

7MS2DM8 is a 7×7 Markovian environment containing food and obstacles that is illustrated in Figure 5-6. This environment has the optimal number of steps to food equal to $\frac{3+4+2+2+2+3+1+1+3+1+1+3+1+1+2+3}{16} = 2.125$ and thus the optimal performance is equal to this value.

```
O O O O O O O
O O . O O . O
O . . . . O O
O . . O O . O
O . F O . . O
O . O . . . O
O O O O O O O
```

Figure 5-6: 7MS2DM8 environment

5.2.1 Learning results of XCS-family animat in environments 5MS2DM2, 6MS2DM3, 7MS2DM6, 7nMS2DM6, and 7MS2DM8

As before the animat is equipped with eight sensors around it to detect the sensory information at each step and also, actuators to move it toward one of the possible eight directions. The brain of animat is one of the XCS-family algorithms to give it the ability of adaptive behavior. We have also tried to test the performance of three new XCS algorithms: XCSG with subsumption, XCSS with subsumption, and XCSSG with subsumption. At this time the test environments are 5MS2DM2, 6MS2DM3, 7MS2DM6, 7nMS2DM6, and 7MS2DM8. To provide a way to understand the generalization ability of XCS-family algorithms better, the average population (of classifiers) sizes of XCS-family are shown for various algorithms in each environment. The comparisons of different XCS-family algorithms in different S2DM environments are presented in Figure 5-7 to Figure 5-36. For the S2DM environments the same set of parameters as maze5 as a maze environment have been used (see Table 4.1 to Table 4.4).

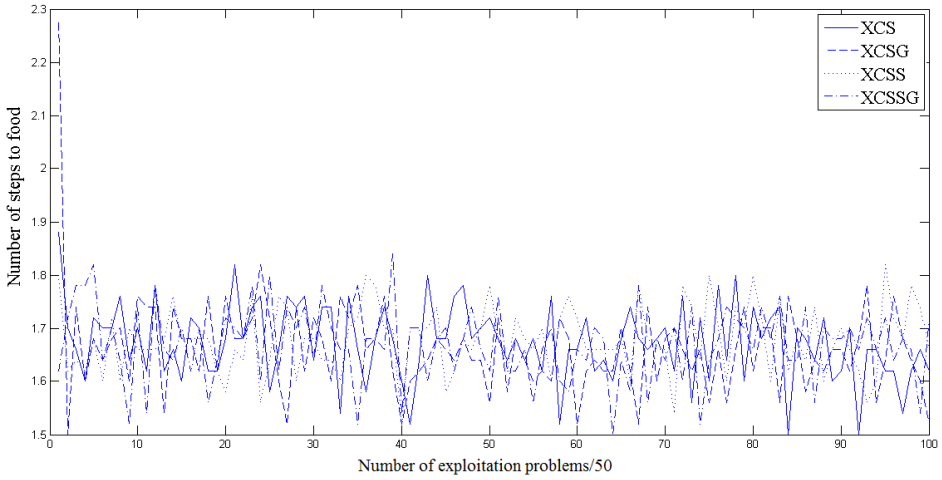


Figure 5-7: Comparison of performance of different XCS algorithms in 5MS2DM2.

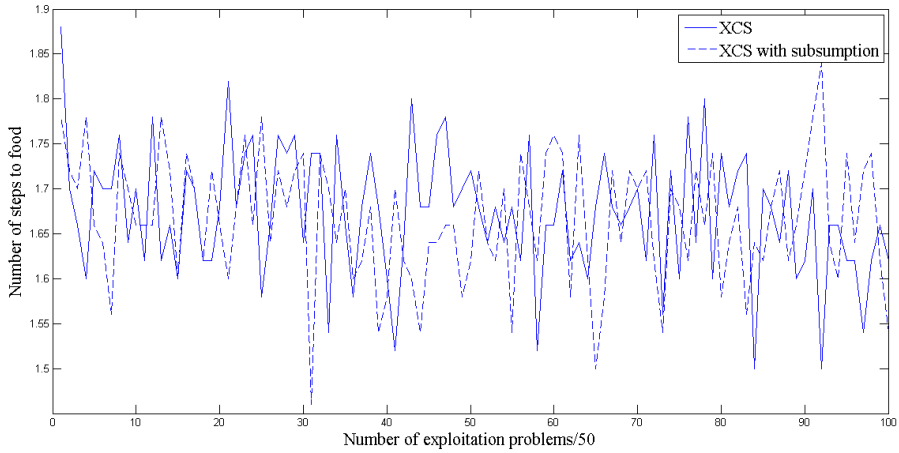


Figure 5-8: Comparison of performance of XCS and XCS with subsumption in 5MS2DM2.

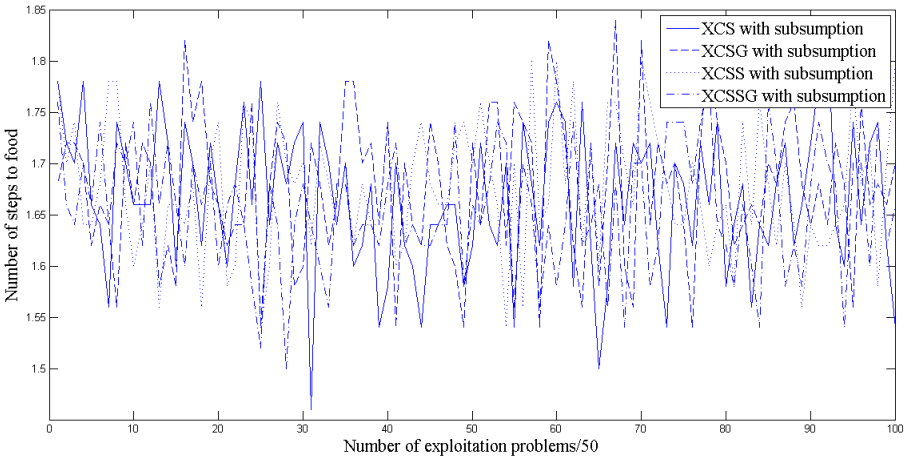


Figure 5-9: Comparison of different XCS algorithms in 5MS2DM2 when the subsumption mechanism is activated.

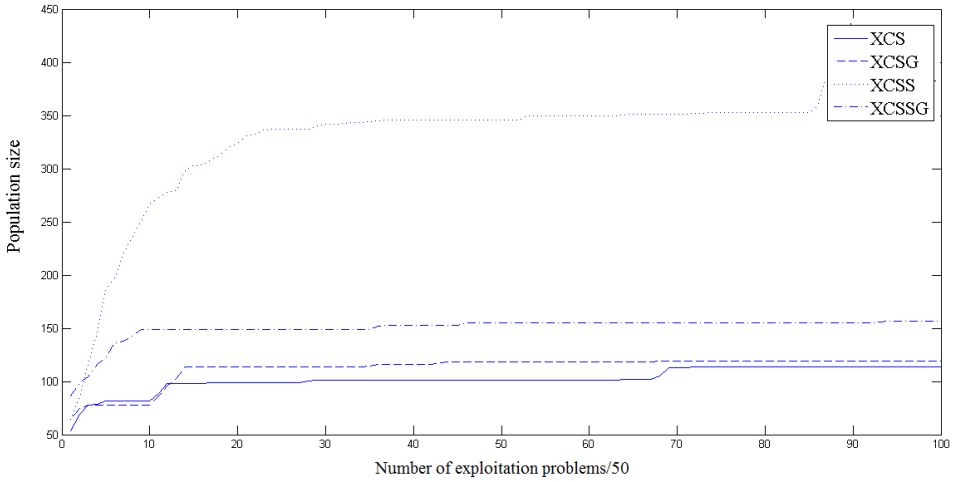


Figure 5-10: Comparison of population of classifiers in XCS, XCSG, XCSS, and XCSSG algorithms in 5MS2DM2.

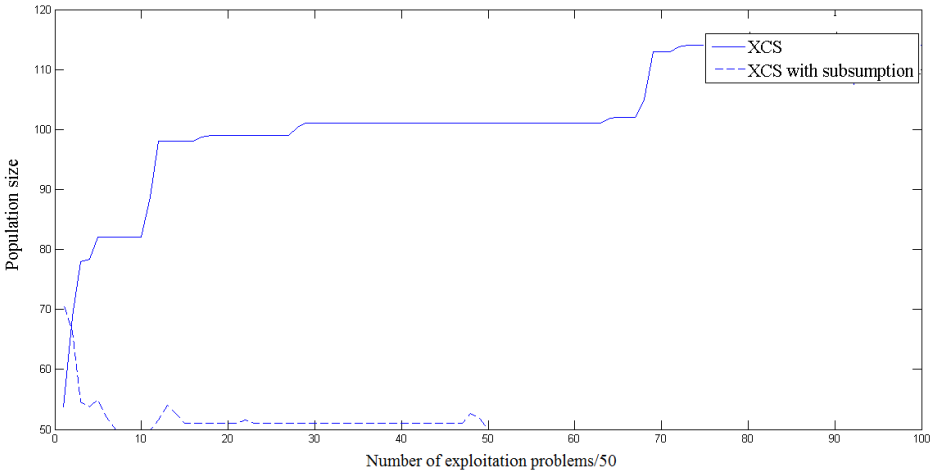


Figure 5-11: Comparison of population of classifiers in XCS, and XCS with subsumption in 5MS2DM2.

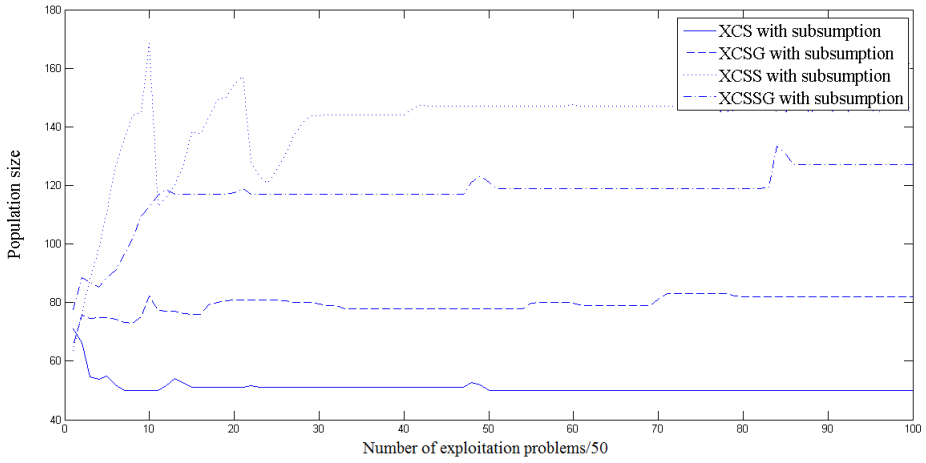


Figure 5-12: Comparison of population of classifiers in XCS-family algorithms with subsumption in 5MS2DM2.

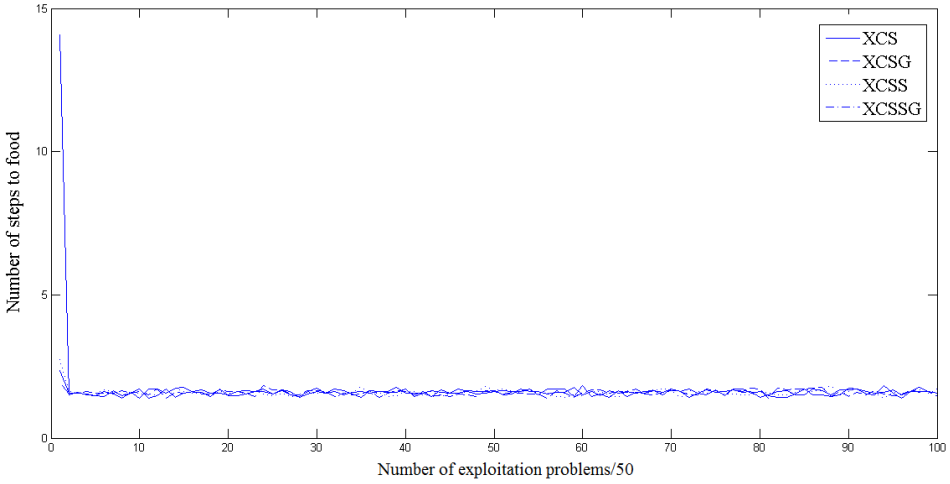


Figure 5-13: Comparison of performance of different XCS algorithms in 6MS2DM3.

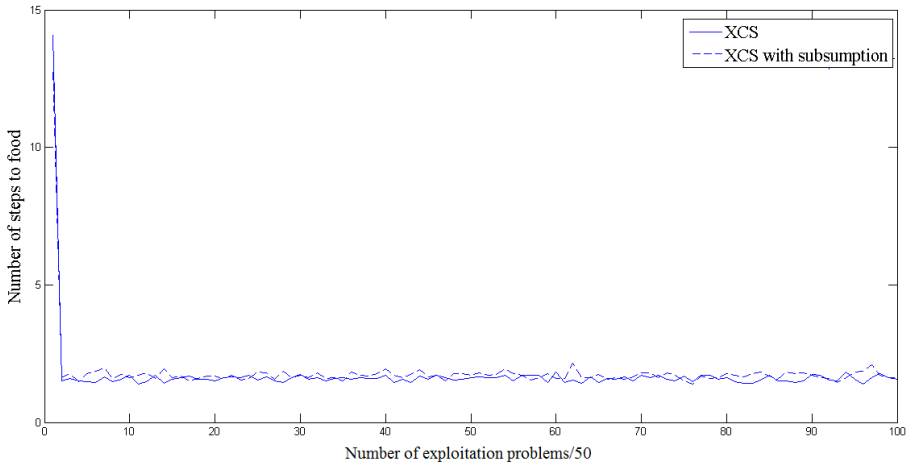


Figure 5-14: Comparison of performance of XCS and XCS with subsumption in 6MS2DM3.

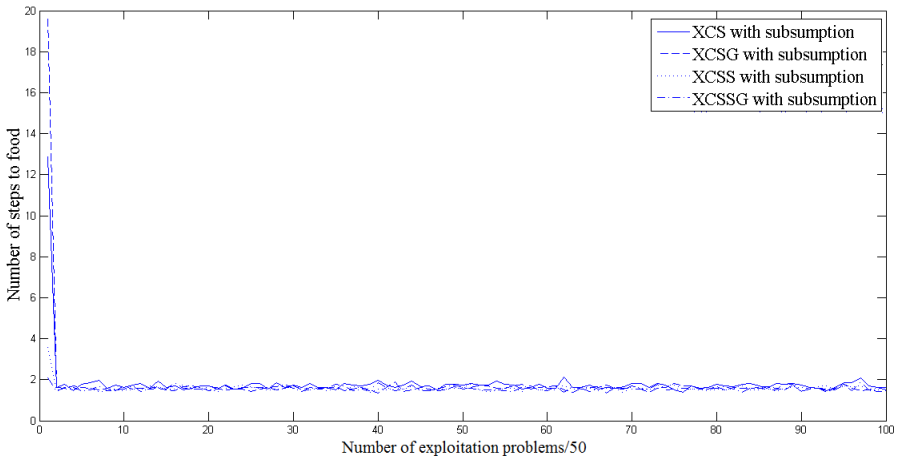


Figure 5-15: Comparison of different XCS algorithms in 6MS2DM3 when the subsumption mechanism is activated.

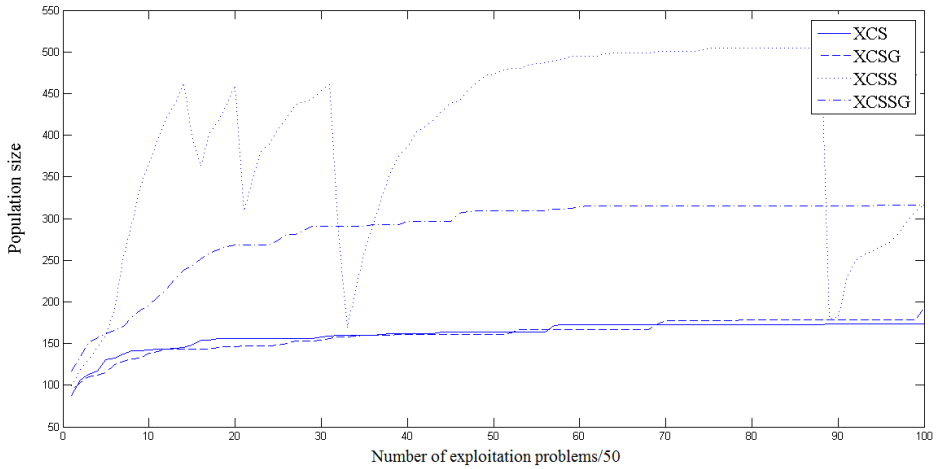


Figure 5-16: Comparison of population of classifiers in XCS, XCSG, XCSS, and XCSSG algorithms in 6MS2DM3.

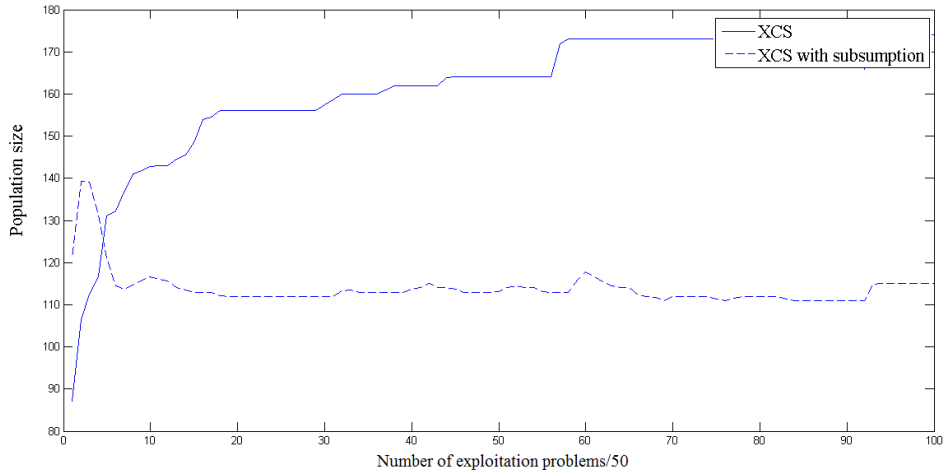


Figure 5-17: Comparison of population of classifiers in XCS, and XCS with subsumption in 6MS2DM3.

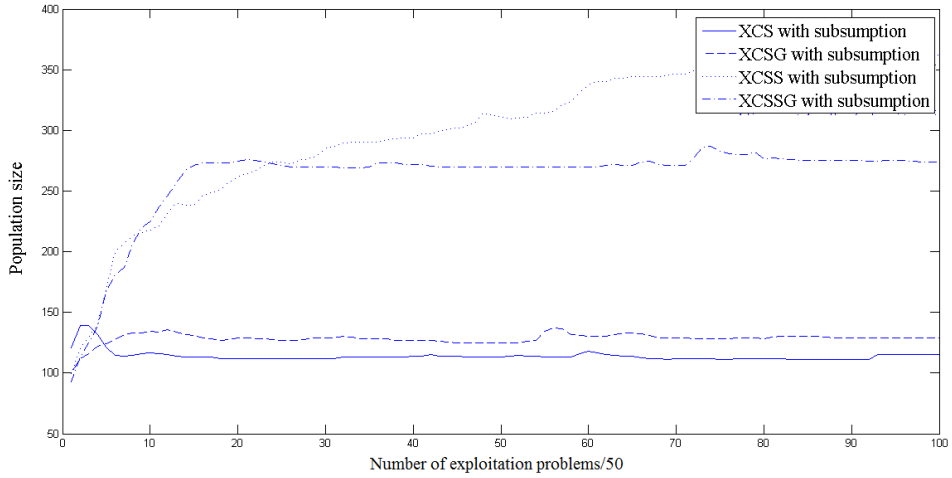


Figure 5-18: Comparison of population of classifiers in XCS-family algorithms with subsumption in 6MS2DM3.

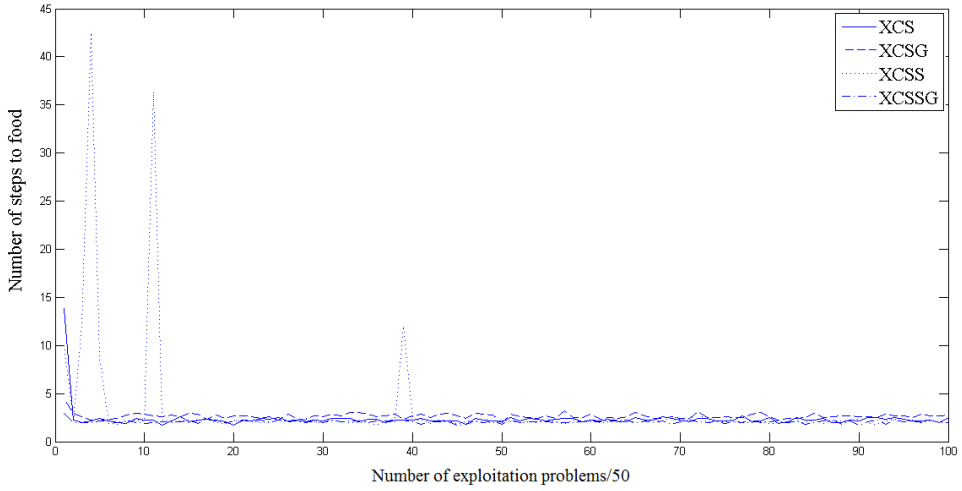


Figure 5-19: Comparison of performance of different XCS algorithms in 7MS2DM6.

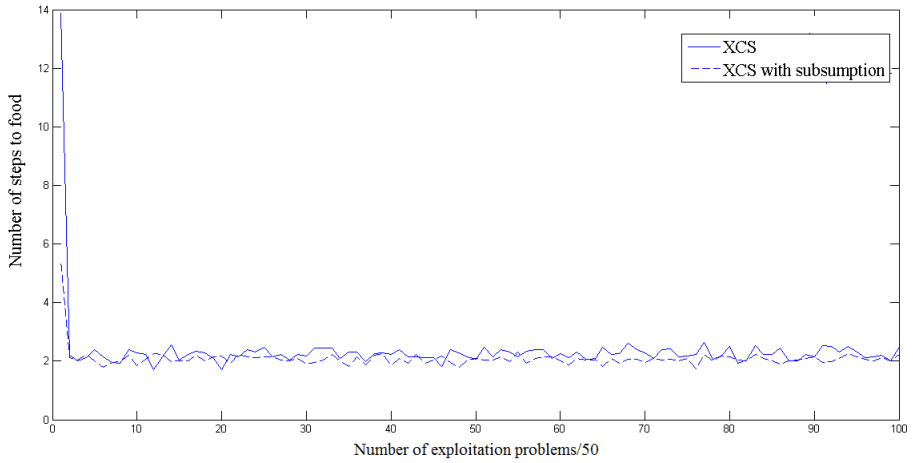


Figure 5-20: Comparison of performance of XCS and XCS with subsumption in 7MS2DM6.

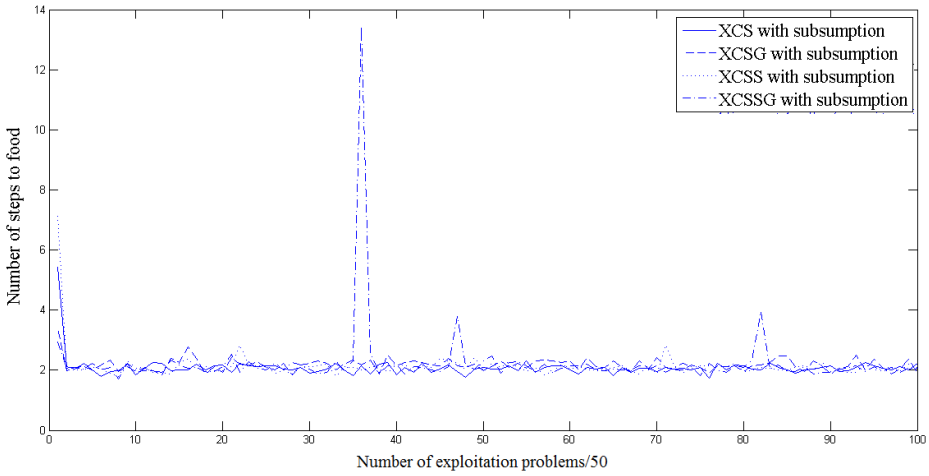


Figure 5-21: Comparison of different XCS algorithms in 7MS2DM6 when the subsumption mechanism is activated.

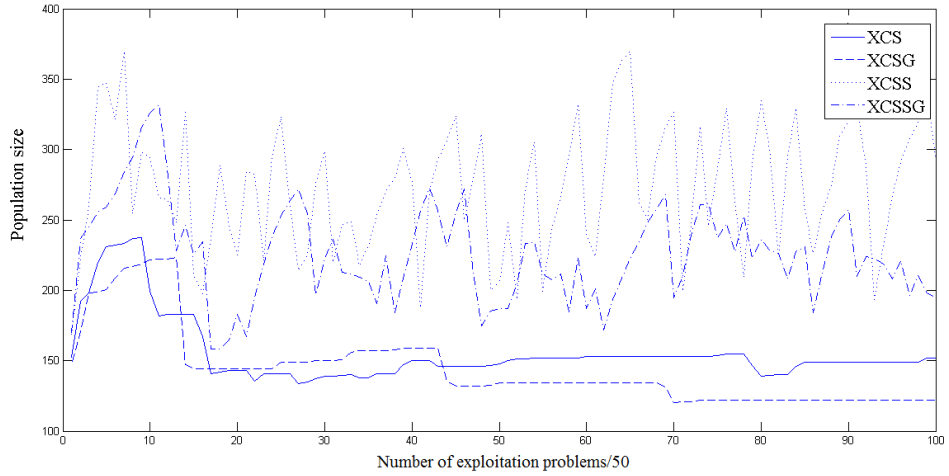


Figure 5-22: Comparison of population of classifiers in XCS, XCSG, XCSS, and XCSSG algorithms in 7MS2DM6.

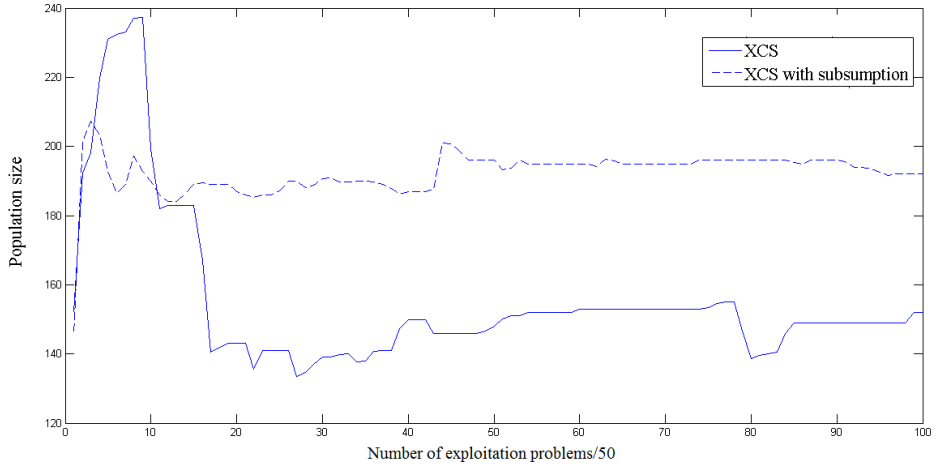


Figure 5-23: Comparison of population of classifiers in XCS, and XCS with subsumption in 7MS2DM6.

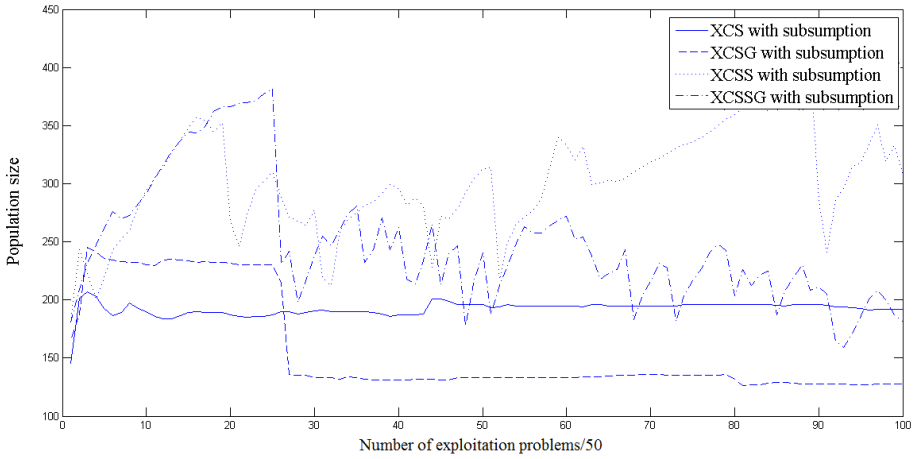


Figure 5-24: Comparison of population of classifiers in XCS-family algorithms with subsumption in 7MS2DM6.

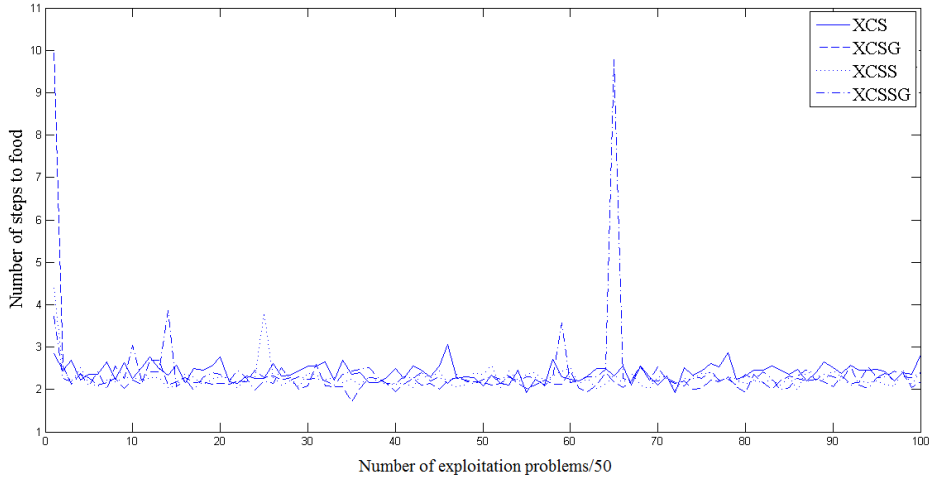


Figure 5-25: Comparison of performance of different XCS algorithms in 7nMS2DM6.

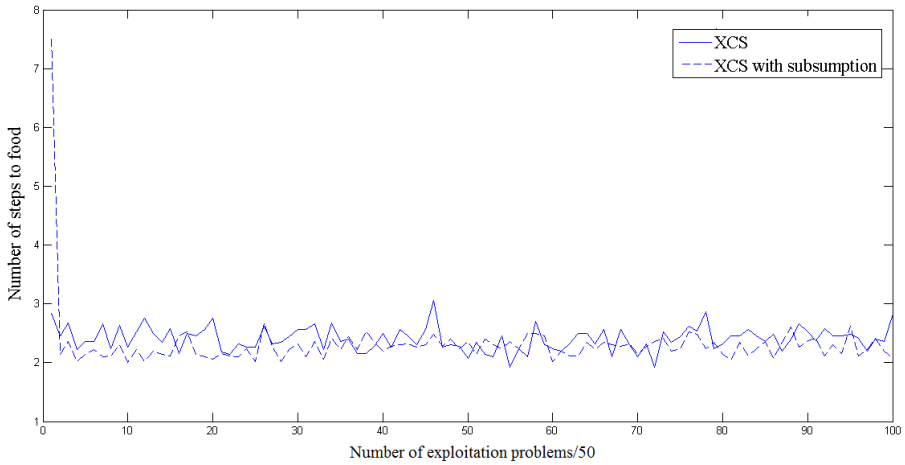


Figure 5-26: Comparison of performance of XCS and XCS with subsumption in 7nMS2DM6.

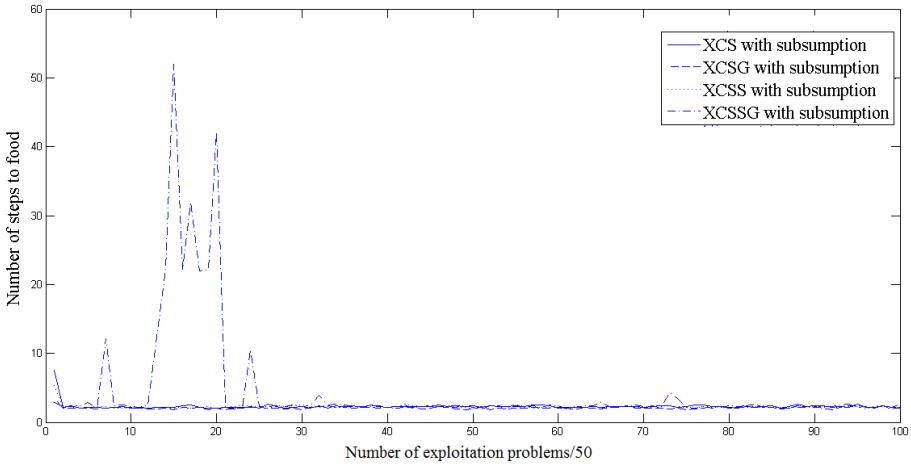


Figure 5-27: Comparison of different XCS algorithm in 7nMS2DM6 when the subsumption mechanism is activated.

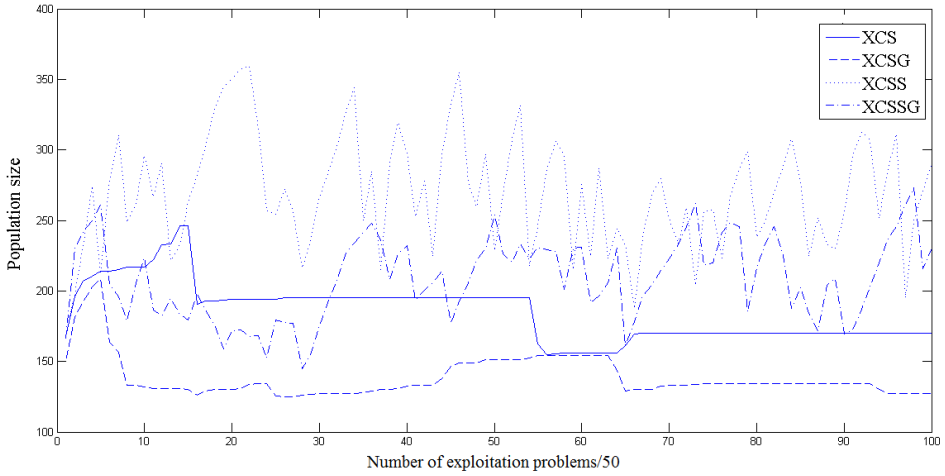


Figure 5-28: Comparison of population of classifiers in XCS, XCSG, XCSS, and XCSSG algorithms in 7nMS2DM6.

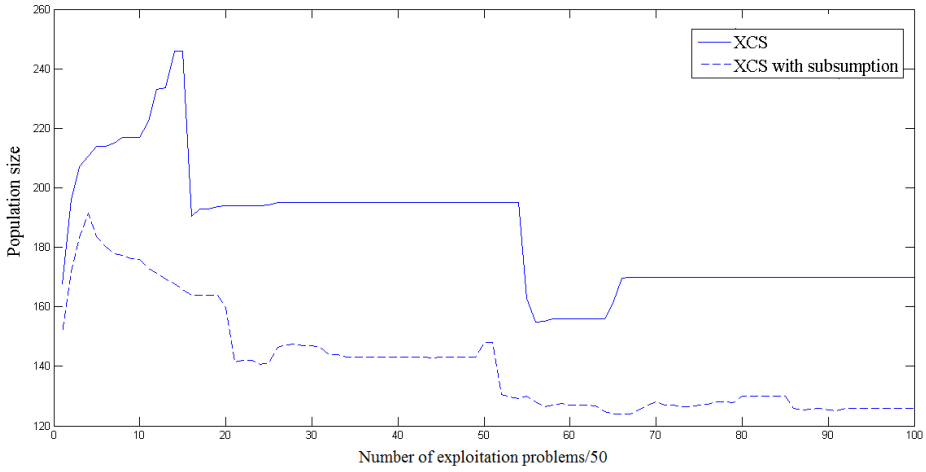


Figure 5-29: Comparison of population of classifiers in XCS, and XCS with subsumption in 7nMS2DM6.

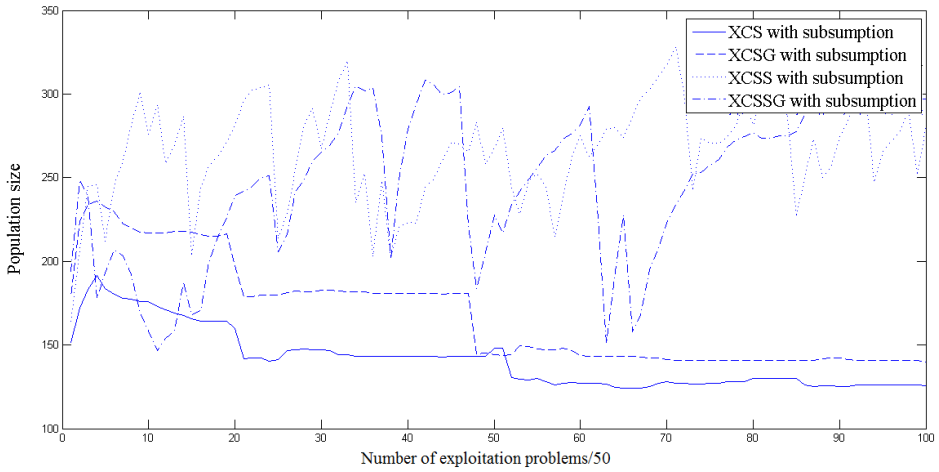


Figure 5-30: Comparison of population of classifiers in XCS-family algorithms with subsumption in 7nMS2DM6.

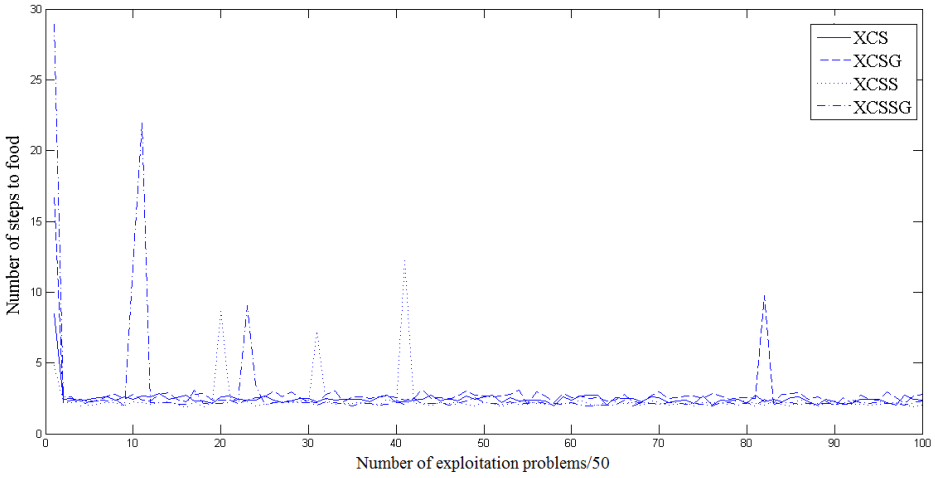


Figure 5-31: Performances of XCS-family algorithms in 7MS2DM8.

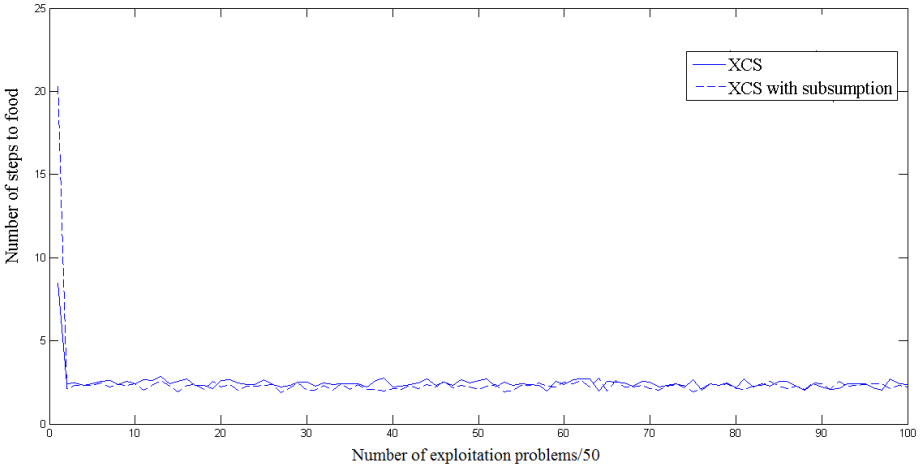


Figure 5-32: Comparison of XCS and XCS with subsumption algorithms 7MS2DM8.

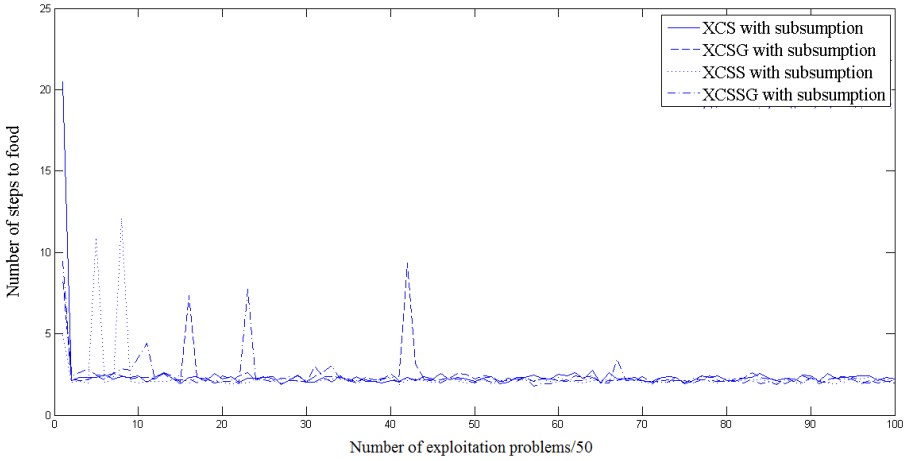


Figure 5-33: Comparison of different XCS algorithm in 7MS2DM8 when the subsumption mechanism is activated.

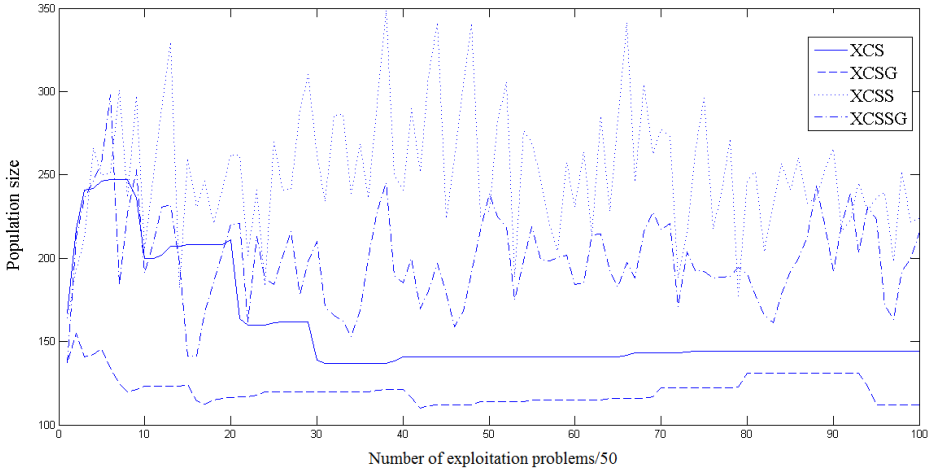


Figure 5-34: Comparison of population of classifiers in XCS, XCSG, XCSS, and XCSSG algorithms in 7MS2DM8.

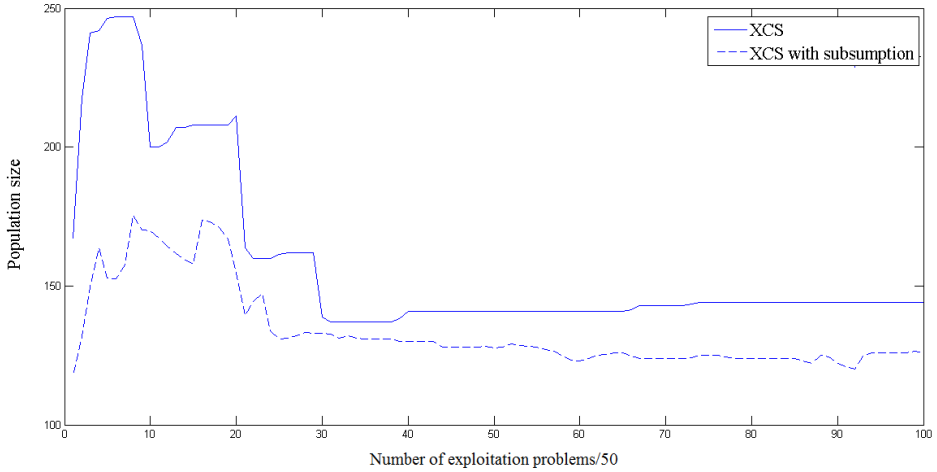


Figure 5-35: Comparison of population of classifiers in XCS, and XCS with subsumption in 7MS2DM8.

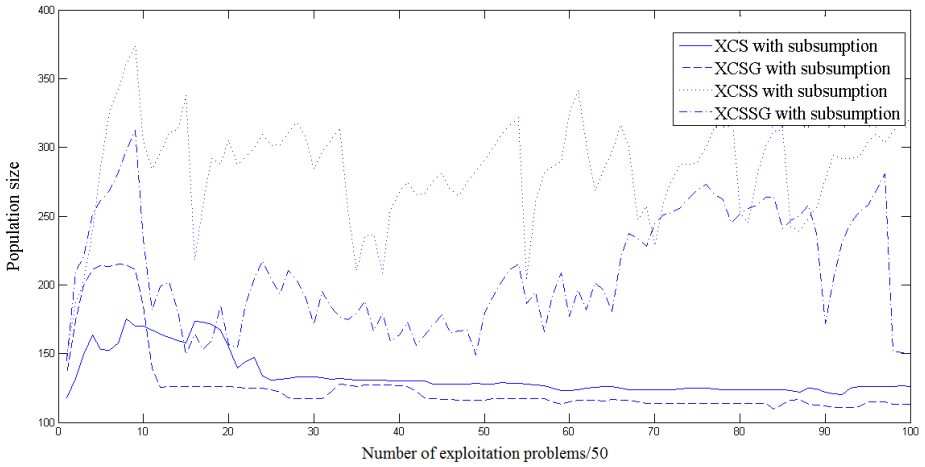


Figure 5-36: Comparison of population of classifiers in XCS-family algorithms with subsumption in 7MS2DM8.

As the results show, in 5MS2DM2, all the XCS-family algorithms learn to reach to the optimal performance that is around 1.66. The variation in the performance curve is because of the short path that animat passes to reach to the food and the average from the starting point is nearly the same. The variation is not also a lot, because the values are magnified.

In 6MS2DM3, and 7MS2DM6 all the XCS-family algorithms can learn to approach to the optimal performance stably and fast except XCSS and XCSSG with subsumption in 7MS2DM6 that are somehow slower than the other methods and at some situations may fail to reach to food as fast as the optimal number of steps to food. It is obvious from the curves that the optimal performance for 6MS2DM3 is around 1.58 and for 7MS2DM6 it is around 2.05. These values are the optimal performances which were proven theoretically before. So, it shows that the animat completely learns to reach to the average optimal number of steps to food.

In non-Markovian environment 7nMS2DM6, the XCS classifier system can learn but as it is obvious from the curve the variation in performance is not very stable as the other three environments. It is because of similar sensory information that the system receives from the environment and may make mistake in choosing the best action for that situation. However, since the environment is not big and also, the distance between the similar cells with the same sensory information is not a lot, the number of step to food is near to stable value in comparison to the other Markovian environments that the performances reach to stable value. The optimal performance for 7nMS2DM6 environment is around 2.22 and the system approaches to a value

close to 2.22. For 7nMS2DM6, using XCSSG with subsumption is not suggested because it is somehow slow and at the first starts with higher values of steps to food.

In 7MS2DM8 the results show that XCS algorithm can easily learn to approach to the optimal performance in this random environment. Comparison of various XCS algorithms show that XCS, XCS with subsumption, and XCSG are very close and all approach to the optimal performance that is 2.125. The results of XCSS, XCSSG, XCSS with subsumption, and XCSG with subsumption also approaches to the optimal performance but in some points there are some abnormal values that are created because of the existence of *Specify* operator in environments that no over-general classifier is produced or use the subsumption mechanisms that removes some important classifiers from the population.

According to the obtained results, we can conclude that in environments where over-general classifiers are not generated, it is better to use simple XCS or XCS with subsumption but in environments such as maze5 that over-general classifiers are created using XCSS and XCSSG (*specify* operator) improves the performance. Thus:

- As a way to start learning (a recipe), it is better to first start learning by simple XCS, if it approaches to the optimal performance there is no need to use other methods, but if it doesn't approach to the optimal performance we can continue learning by the other XCS-family algorithms.
- Using subsumption with XCSS or XCSG with and also XCSSG can be removed from the check list to be tested as a XCS-family algorithm on any kind of environment.

The analysis of generalization in XCS-family algorithms using the number of classifiers in the population set: The results of change in the average number of the classifiers in the population for each environment are presented and are compared for various XCS-family algorithms. By the analysis we can achieve the following conclusions about the operation of each XCS-family algorithm:

- After several problems the number of classifiers reaches to nearly a fixed value that shows the generalization ability of XCS-family algorithms. For environments such as 7MS2DM6 the generalization ability is more clear: the number of classifiers first increases because at the first the system needs to generate classifiers that are matched

with the sensory input from the environment and after a number of steps the number of classifiers decreases because the system tries to remove the classifiers that are less general and only keep ones that are useful and prepares the minimal population of accurate classifiers. So, only the classifiers that are general enough and accurate are kept in the system. It shows the generalization ability of the system.

- The subsumption mechanism often decreases the number of classifiers to give the system higher degree of generalization ability. However, this mechanism sometimes removes classifiers that are important for the system and fails in some situation, and yields decrease in the performance of the system.
- XCS and XCSG are close in generalization ability (XCSG is a little bit more powerful), but XCSS generates the higher number of classifiers in comparison to the other methods, and this is the mechanism that XCSS tries to overcome the complexity of the environment by generating higher number of classifiers and to create a more detailed mapping. The oscillation in the number of classifiers in XCSS and XCSSG are very clear that is because of the *Specify* operator which is used in these kinds of classifier systems.
- XCSSG has a better ability of generalization than XCSS. So, in generalization it is the improved version of XCSS.

5.3 Unstable resource problem with XCS-animat

To study the ability of XCS animat to tackle with the problems in which the environment is changing, an unstable resource problem is designed. Unstable resource problem is the problem of an animat that tries to reach to a moving food. According to definition of Wilson's animat we can adopt unstable resource problem by trying to investigate the learning ability of animat when the place of food changes to one of the neighbouring cells. For this experiment, 7MS2DM6 is considered as the test environment. The animat learns in 7MS2DM6 in 7500 problems and suddenly the food moves to direction 1 (among 0-7). At this situation the animat experiences a new environment which we call it 7MS2DM6-B (see Figure 5-56 and Figure 5-38) with the average number of steps to food equal to $2.94 \cong 3$. Based on the classifiers that the animat has obtained during learning 7500 problems, in problem 7501, the animat expect to reach to food at

the previous place of food, but when the animat arrives at that cell, it finds no food and also no reward. Therefore, the animat again tries to explore the environment and obtain new sensory information about the environment and change some of classifiers in the population set and creates new ones. So, after 7500 steps the animat adapts to the new situation.

This problem is tested on 7MS2DM6 and the results are presented in Figure 5-39 to Figure 5-42.

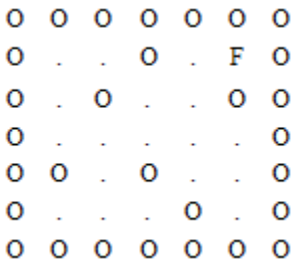


Figure 5-37: 7MS2DM6-B environment

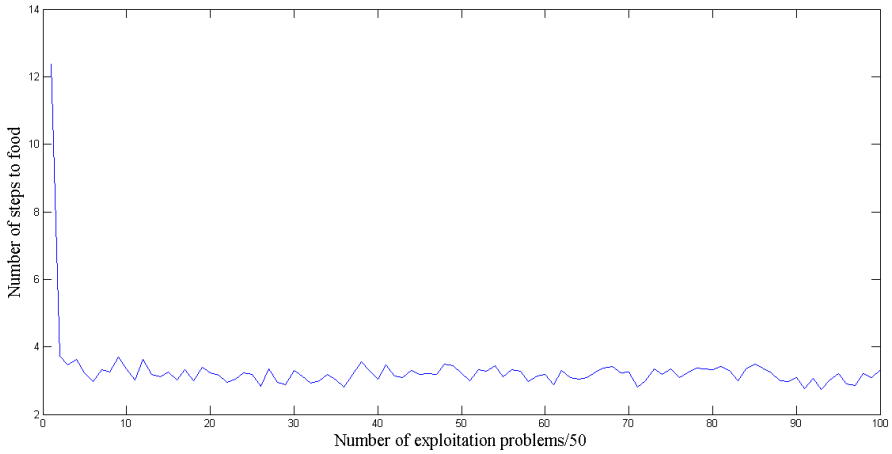


Figure 5-38: Learning in 7MS2DM6-B environment and the optimal performance.

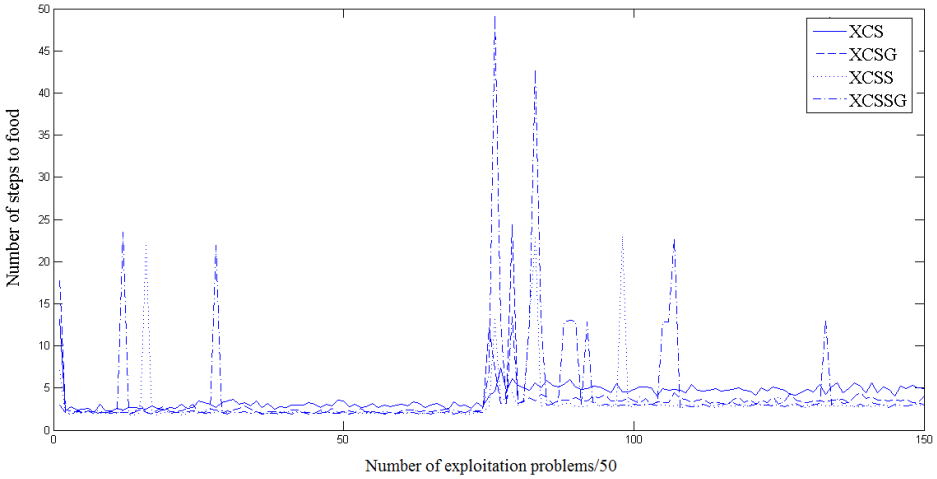


Figure 5-39: Unstable resource problem in 7MS2DM6 with different XCS-family algorithms when the food moves toward direction 1.

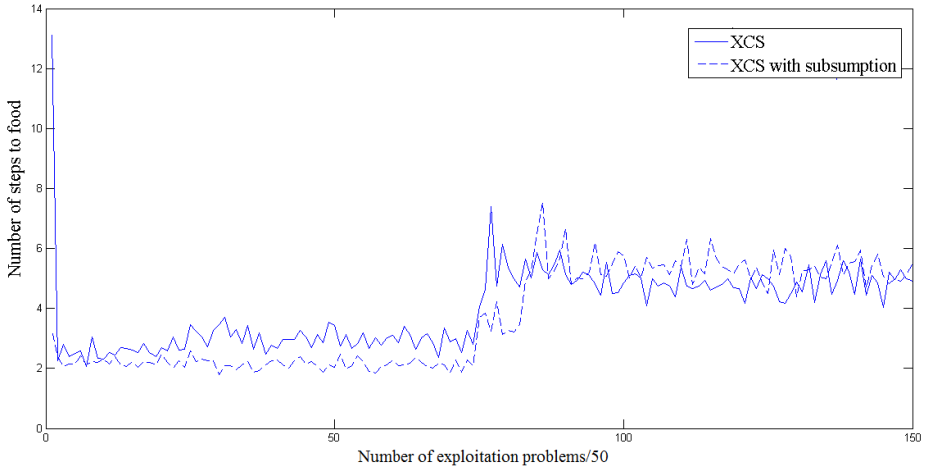


Figure 5-40: Unstable resource problem in 7MS2DM6. Comparison between XCS and XCS with subsumption when the food moves toward direction 1.

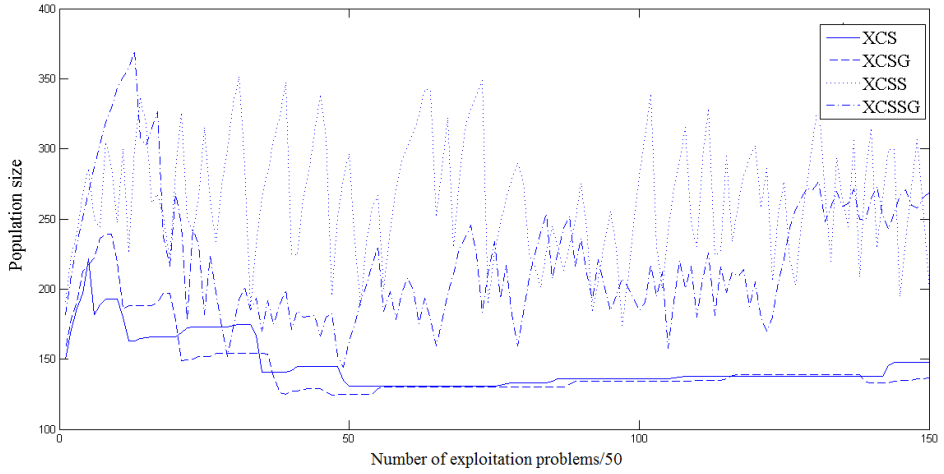


Figure 5-41: Unstable resource problem in 7MS2DM6. Comparison of population sizes.

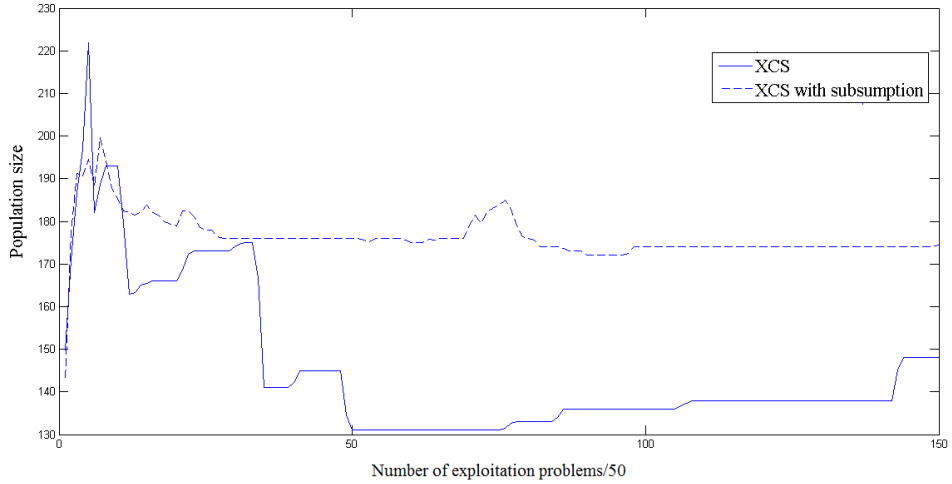


Figure 5-42: Unstable resource problem in 7MS2DM6. Comparison of population size between XCS and XCS with subsumption.

The results of learning show that the best algorithm to tackle with the XCS unstable resource problem is XCSG. XCSG animat at the first reaches to the optimal performance of 7MS2DM6, and after change in the place of food adapts to the new situation and approaches to the optimal performance of 7MS2DM6-B rapidly. XCS and XCS with subsumption at the first approach to the optimal performance but after change in the place of food, they cannot reach to the optimal value of 7MS2DM6-B that is around 3, instead they approach to a value around 5. XCSS and

XCSSG at the first approach to the optimal performance of 7MS2DM6 but with some picks during learning, and although after change in the place of food they approach to a value around 3, they again have picks during learning which is because of the *Specify* operator.

The analysis show that at point 7500 the number of steps to food and also the number of classifiers in the population increases and this is exactly what we had expected before, because the situation of food has changed and the animat produces new classifiers to adapt with the new situation and get rid of some of the previous classifiers that are not useful for the current situation. This procedure leads to increase in the number of steps to food after around 7500 problems; because time is needed for animat to explore and exploit the new situation and to be adapted to the new place of food.

5.4 Interspecific competition problem and XCS animat

In this section a scenario is designed to study the competitive behavior of an ecosystem of XCS-family animats for resources that is called interspecific competition. In this scenario, two kinds of animats are considered: XCS animats and XCSSG animats. At each step only one animat exists in the environment but that animat type is chosen based on a competition that holds among XCS animats and XCSSG animats. The competition between two population types is based on competitive Lotka-Volterra equations. An animat type is chosen according to a probability which is proportional to the size of each population.

5.4.1 Competitive Lotka-Volterra equation

Competitive Lotka-Volterra equation is a non-linear differential equation that describes the population dynamics in an environment when two species are in competition for a common resource (Interspecific competition) [80]. The structure of a community of species is determined by the dynamics of interaction between the species. In addition to the interaction between individuals of different species, the interaction between different individuals in one species can affect the population dynamics of the community. The equation for dynamics of the population growth of species 1 (X) and species 2 (Y) are as follows:

$$\frac{dX}{dt} = r_X X \left[\frac{K_X - X - \alpha_{XY} Y}{K_X}\right] \qquad (5.1)$$

$$\frac{dY}{dt} = r_Y Y \left[\frac{K_Y - Y - \alpha_{YX} X}{K_Y}\right] \quad (5.2)$$

X and Y are the population size of species 1 and species 2.

r_X and r_Y are the intrinsic growth rate of species 1 and species 2.

K_X and $K_Y$are the carrying capacities of species 1 and species 2 when the other species is absent.

α_{XY} and α_{YX} are the effect of one species on the growth of the other species. They represent for example how many individual of species 1 equal to species 2. $\alpha_{XY} > 0$ and $\alpha_{YX} > 0$ shows the competition between the species.

For Compettitive Lotka-Volterra equation the equilibrium points where the change in the population is zero are as followed:

$\frac{dX}{dt} = 0$ and $\frac{dY}{dt} = 0$: four equilibrium points are obtained. For three of them one or both the species are absent. Only for one of these equilibrium points both the species are available which is $X = \frac{K_X - \alpha_{XY} K_Y}{1 - \alpha_{XY}\alpha_{YX}}$ and $Y = \frac{K_Y - \alpha_{YX} K_X}{1 - \alpha_{XY}\alpha_{YX}}$.

5.4.2 XCS-XCSSG competition

According to the definition of Wilson's animat problem, at each step only one animat exists in the environment that tries to explore and exploit the environment for food. To use competitive Lotka-Volterra equation in the context of Wilson's animat problem, a pool is considered which contains population of two types of animats: XCS animats and XCSSG animats. The population compete based on the competitive Lotka-Volterra equation and the winner species at each time-step is selected to perform on the environment. The number of steps to food for the selected animat is used as a feedback for the system to update parameters of the competitive equation. Probability of choosing a species (XCS or XCSSG) is proportional to the percentage of its population in the pool (see Figure 5-43). Intrinsic growth rate and the carrying capacities are kept constant and the parameters of competition are affected by the number of steps to food.

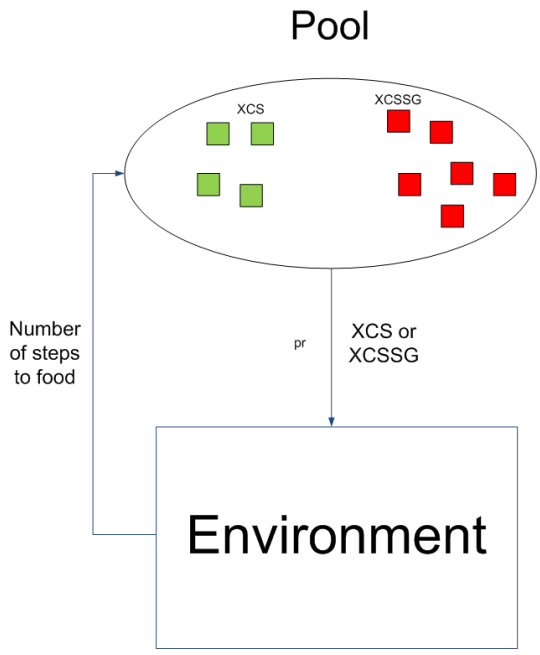


Figure 5-43: Competition of XCS and XCSSG animats for learning to find food in the environment.

As it was presented the competitive Lotka-Volterra equation is as follows

$$\frac{dX}{dt} = r_X X\left[\frac{K_X - X - \alpha_{XY} Y}{K_X}\right], \tag{5.3}$$

$$\frac{dY}{dt} = r_Y Y\left[\frac{K_Y - Y - \alpha_{YX} X}{K_Y}\right]. \tag{5.4}$$

The discrete form for this equation to update the population is

$$X_{n+1} = X_n + r_X X_n\left[\frac{K_X - X_n - \alpha_{XY} Y_n}{K_X}\right], \tag{5.5}$$

$$Y_{n+1} = Y_n + r_Y Y_n\left[\frac{K_Y - Y_n - \alpha_{YX} X_n}{K_Y}\right]. \tag{5.6}$$

To update the parameters α_{XY} and α_{YX} at each step by inspiration from standard Widrow-Hoff delta rule [1] with learning parameter α the following equations will be obtained,

when XCS is chosen

$$\alpha_{XY}^{(n+1)} = \alpha_{XY}^{(n)} + \alpha\left(P_X - \alpha_{XY}^{(n)}\right), \tag{5.7}$$

$$\alpha_{YX}^{(n+1)} = \alpha_{YX}^{(n)} - \alpha\left(P_Y - \alpha_{YX}^{(n)}\right), \tag{5.8}$$

When XCSSG is chosen:

$$\alpha_{XY}^{(n+1)} = \alpha_{XY}^{(n)} - \alpha\left(P_X - \alpha_{XY}^{(n)}\right), \tag{5.9}$$

$$\alpha_{YX}^{(n+1)} = \alpha_{YX}^{(n)} + \alpha\left(P_Y - \alpha_{YX}^{(n)}\right), \tag{5.10}$$

Where P_X and P_Y are defined as:

$$P_1 = \frac{(NSF_n - NSF_{n-1})}{\frac{NSF_n + NSF_{n-1}}{2}}\left(\frac{K_X}{2r_X}\frac{X_n - X_{n-1}}{X_n Y_n}\right), \tag{5.11}$$

$$P_2 = \frac{(NSF_n - NSF_{n-1})}{\frac{NSF_n + NSF_{n-1}}{2}}\left(\frac{K_Y}{2r_Y}\frac{Y_n - Y_{n-1}}{X_n Y_n}\right). \tag{5.12}$$

Lower α_{XY} increases the population of X and lower α_{YX}increases the population of Y that are desirable for each population (to increase its probability of selection). So, Equations (5.8) and (5.9) are added to punish in the sense that one of the species are not chosen (note that the desire value for $(NSF_n - NSF_{n-1})$ is negative or equal to zero and the negative sign behind α makes an undesirable value which can be considered as a punishment). NSF is the abbreviation for the "number of steps to food".

At each step one of two species of XCS or XCSSG are chosen proportional to their percentage in the population which is:

$$pr_n(X_n) = \frac{X_n}{X_n + Y_n} \tag{5.13}$$

$$pr_n(Y_n) = \frac{Y_n}{X_n + Y_n} \tag{5.14}$$

The probability will be updated when the population of species in the pool changes.

5.4.3 Experimental results

To perform experiment in the proposed platform the carrying capacities of both species for each environment are assumed fixed and equal to the number of blank points in the environment because carrying capacity is equal to the maximum number of a species in the environment. The intrinsic growth rates for both species are set to 0.2. The initial values for X and Y are supposed

to be 10 and initial values for α_{XY} and α_{YX} equal to 0.1. The value of α is set to 0.001 to keep α_{XY} and α_{YX} positive and lead to a competitive behavior. The experiments perform on 5MS2DM2, 6MS2DM3, 7MS2DM6, and 7nMS2DM6. The results are presented in Figure 5-44 to Figure 5-55.

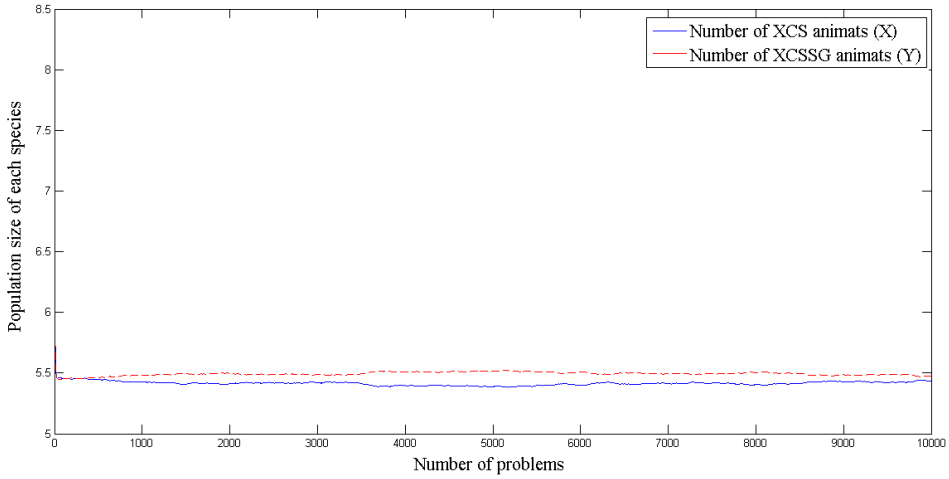


Figure 5-44: Change in the population size of two species in 5MS2DM2.

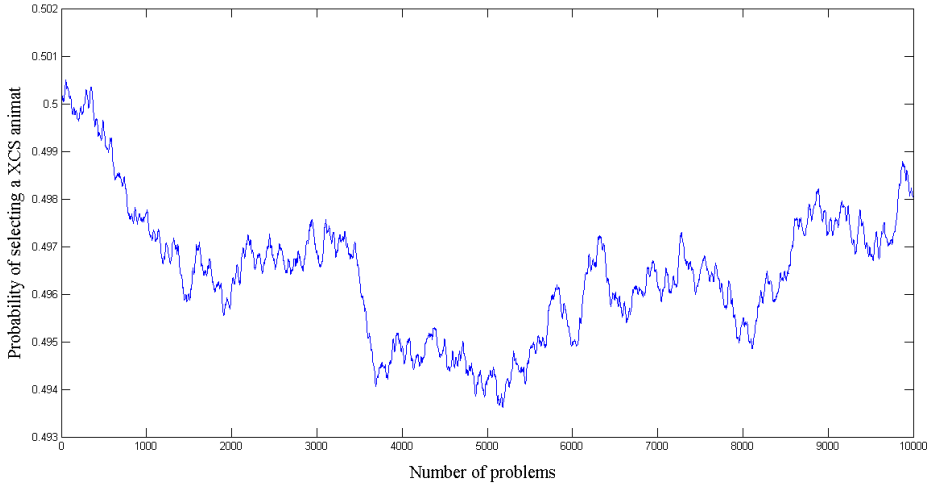


Figure 5-45: Probability of selecting a XCS animat from the pool in 5MS2DM2.

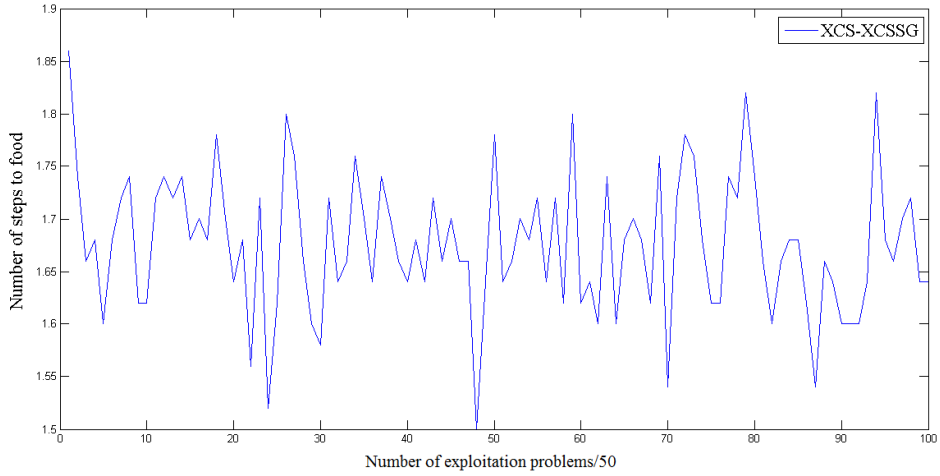


Figure 5-46: Performance of a competitive behavior of XCS-XCSSG classifier systems in 5MS2DM2 environment.

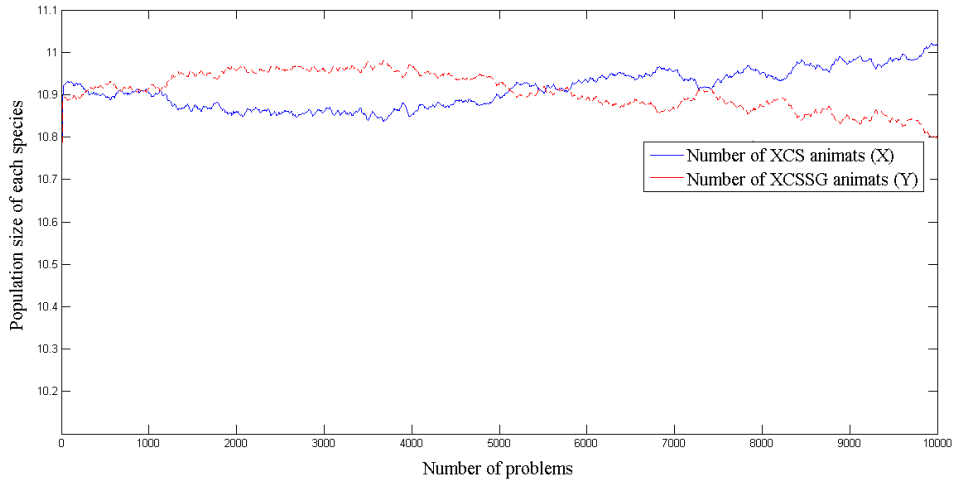


Figure 5-47: Change in the population size of two species in 6MS2DM3.

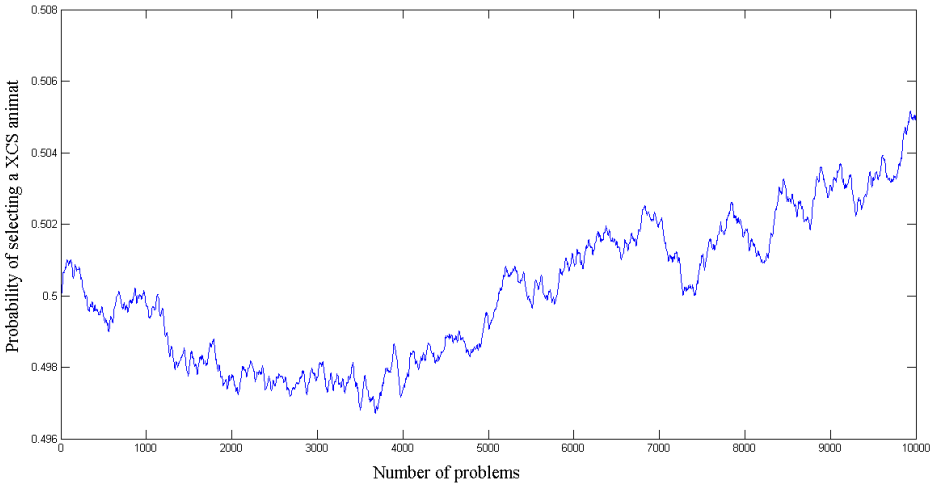


Figure 5-48: Probability of selecting a XCS animat from the pool in 6MS2DM3.

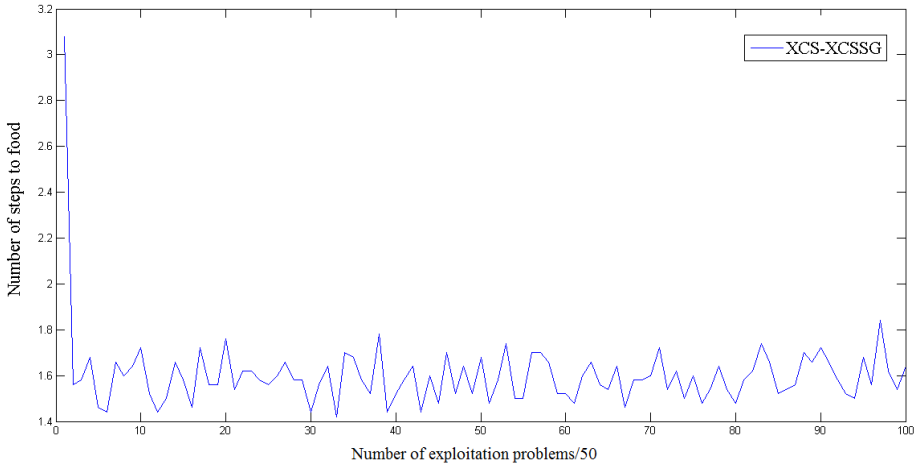


Figure 5-49: Performance of a competitive behavior of XCS-XCSSG classifier systems in 6MS2DM3 environment.

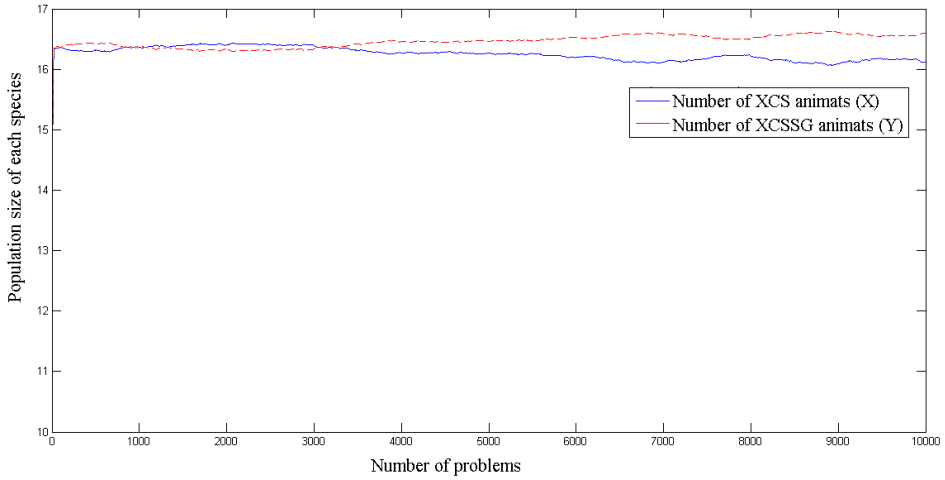


Figure 5-50: Change in the population size of two species in 7MS2DM6.

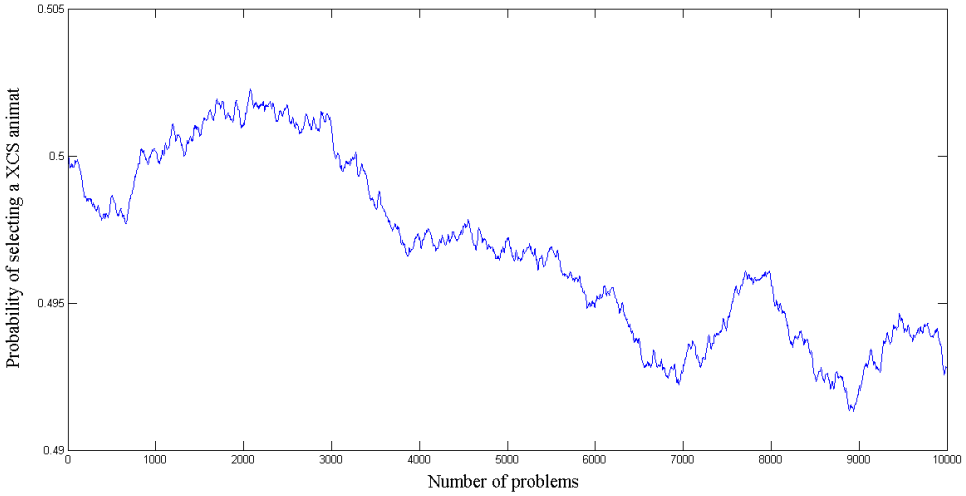


Figure 5-51: Probability of selecting a XCS animat from the pool in 7MS2DM6.

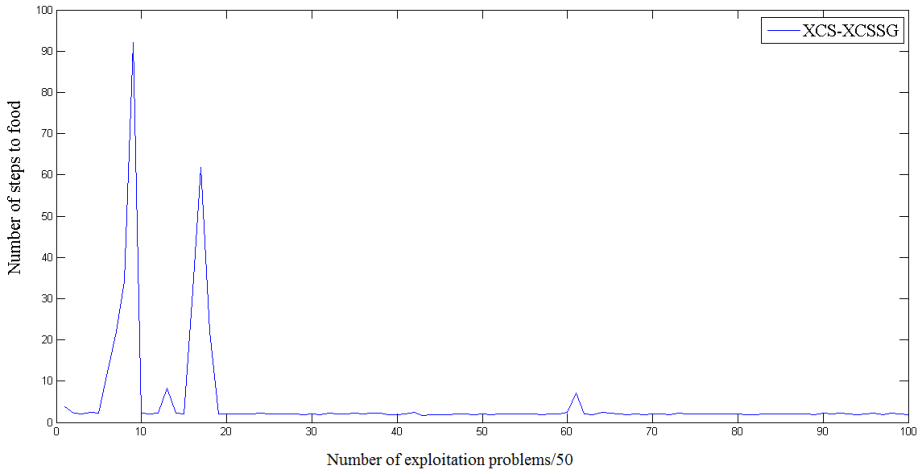


Figure 5-52: Performance of a competitive behavior of XCS-XCSSG classifier systems in 7MS2DM6 environment.

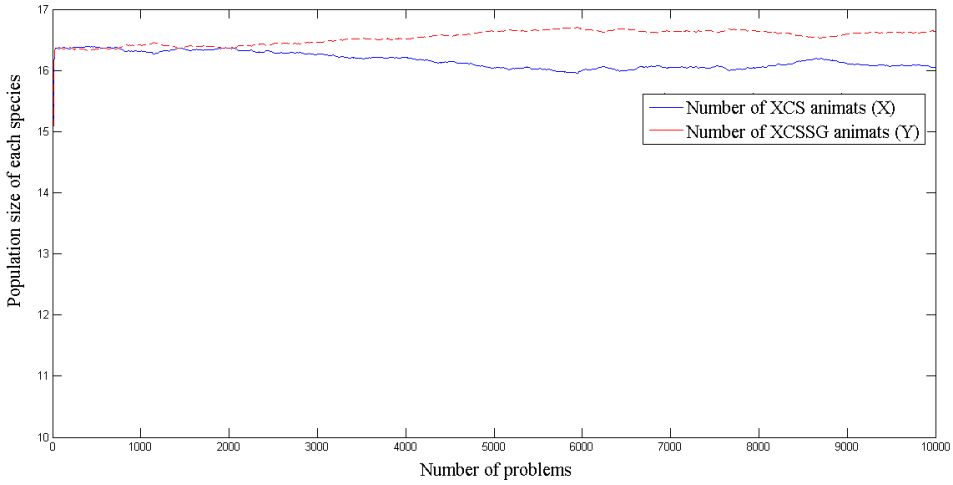


Figure 5-53: Change in the population size of two species in 7nMS2DM6.

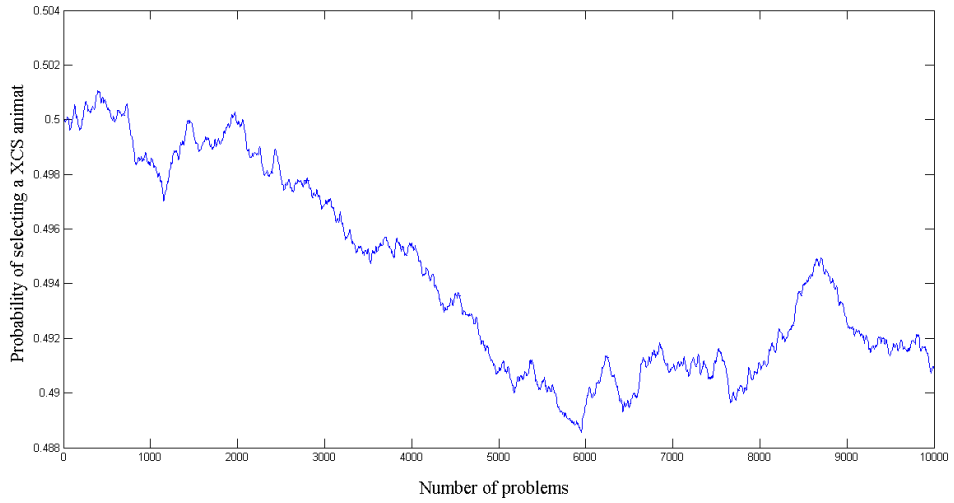


Figure 5-54: Probability of selecting a XCS animat from the pool in 7nMS2DM6.

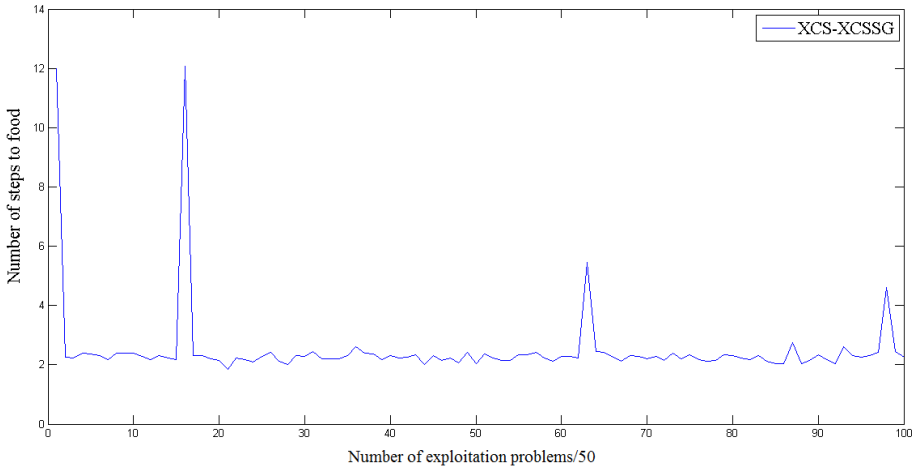


Figure 5-55: Performance of a competitive behavior of XCS-XCSSG classifier systems in 7nMS2DM6 environment.

The results show that the population size of XCS and XCSSG animats reach to approximately the same values as we expected before because the performance of XCS and XCSSG are close for the assumed environments. The probability doesn't have high change because the values of X_n and Y_n change in approximately the same manner and also there is not high difference in the performance of XCS and XCSSG in the considered environments. The number of steps to food works like the previous experiments and reach to the optimal value by using a probabilistic combination of XCS and XCSSG.

5.5 An animat with higher vision abilities

An environment can be Markovian for an animat and be non-Markovian for the other one depending on the sensory information that the animat receives at each step. In the traditional definition of Wilson's animat, the visual abilities are defined in a way that animat only sees eight surrounding cells. This level of visual ability makes many patterns of situating food and obstacles non-Markovian environments. To give animat the ability of making better decision in more complex and higher range of environments, we define the sensory ability of animat in a way that it sees more than eight surrounding cells. So, by this kind of definition many environments that were non-Markovian before will be Markovian environments for this animat. In fact, the animat will have the information of more cells and can recognize its place better and distinguish among cells with the same one-steps sensory information (eight surrounding cells) but with different two-steps sensory information (in addition to the eight surrounding cells, some cells with two steps distance from the position of animat are considered).

To implement this kind of animat, the environment generator is developed to create an environment with several cells of the same one-step sensory information but without any same cells with two-steps sensory information. An additional layer of obstacles are added to the outer layer of maze to provide animat the two-step sensory information at situations where the animat is close to the environment boundary. Two types of two-steps sensory information are assumed: 24 cells and 10 cells (see Figure 5-56 and Figure 5-57). The classifiers in the classifier system in this kind of animat problem are strings of 48 bits and 20 bits instead of 16 bits and also the matrices of sensory information are composed of 10 and 24 objects instead of 8 objects. The actions of animat are kept the same as before which are one step-move toward one of the possible directions into one of the surrounding cells.

22	23	8	9	10
21	7	0	1	11
20	6	*	2	12
19	5	4	3	13
18	17	16	15	14

Figure 5-56: 24 cells sensory information.

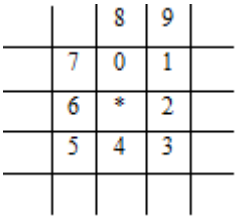


Figure 5-57: 10 cells sensory information.

The environments that are considered for the XCS (and XCSSG) animat with higher vision abilities are named Complex-family environments that are generated by the environment generator. In addition the environment woods101 is also used for the experiment. The environments Complex1, Complex2, Complex3, and Complex4, and also the environment woods101 are presented in Figure 5-58 to Figure 5-62. The same cells in one-step sensory information for Complex-family environments are as follows:

Complex1: 9-31, 25-33, 15-39, 10-24-27-28-32-35-36-42-43, 11-17-14-38-44, 26-34-41, 49-50-51.

Complex2: 13-39, 23-34, 24-35, 26-33, 27-38, 41-45.

Complex3: 25-26.

Complex4: 2-9, 6-21, 7-22.

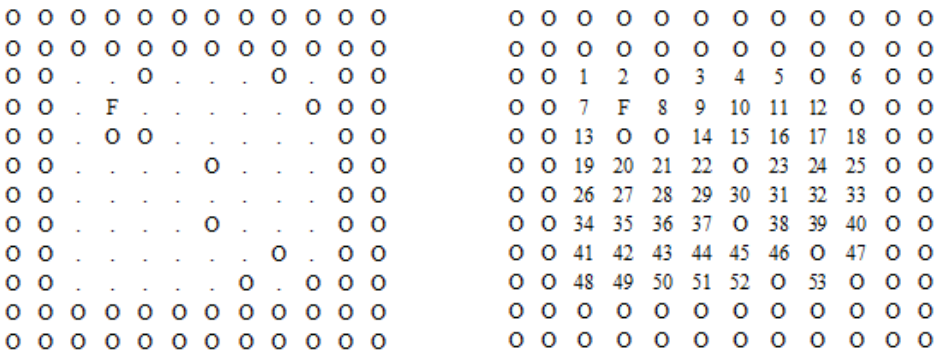


Figure 5-58: The left hand is Complex1 environment. The right hand is Complex1 environment that the blank points are numbered.

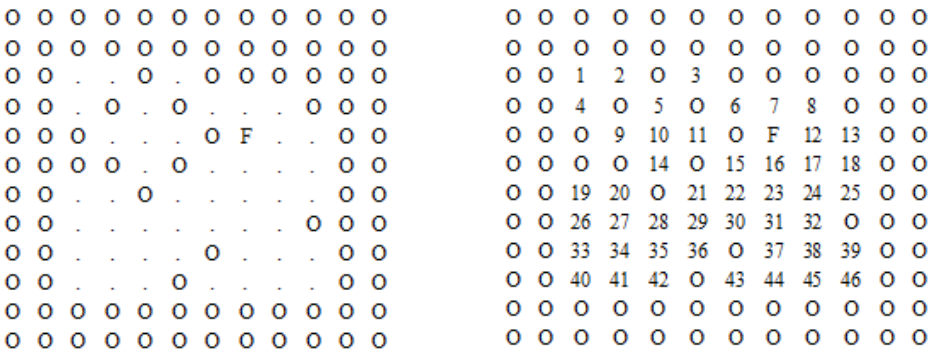


Figure 5-59: The left hand is Complex2 environment. The right hand is Complex2 environment that the blank points are numbered.

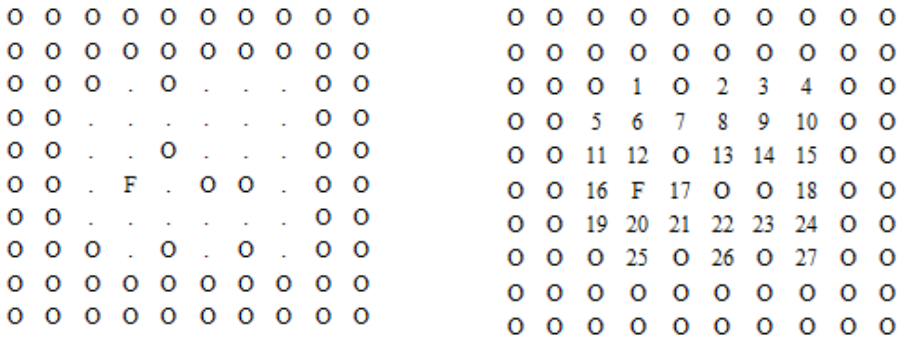


Figure 5-60: The left hand is Complex3 environment. The right hand is Complex3 environment that the blank points are numbered.

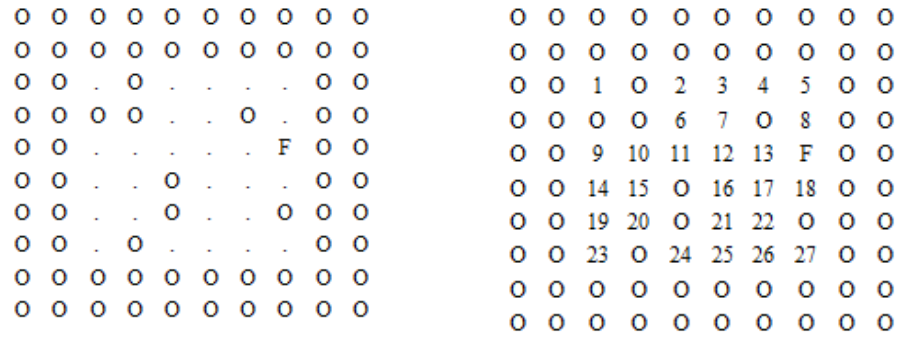


Figure 5-61: The left hand is Complex4 environment. The right hand is Complex4 environment that the blank points are numbered.

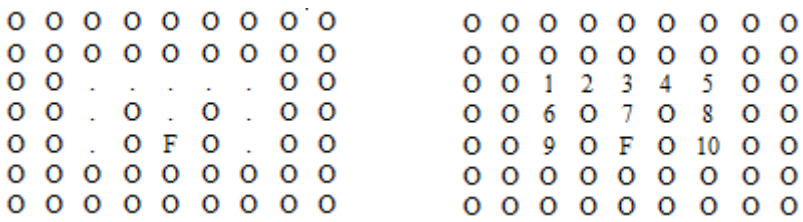


Figure 5-62: The left hand is woods101 environment. The right hand is woods101 environment that the blank points are numbered.

The experiments with the higher vision abilities at Complex1 environment is performed with 24 cells, for Complex2 with 24 cells, for Complex3 with 24 cells, for Complex4 with 10 cells, and for the woods101 with 10 cells of sensory information. The results of learning with simple XCS algorithm and also with XCSSG in each of the Complex-family environments and woods101 with normal sensory abilities and higher vision abilities (24 cells or 10 cells) are presented in Figure 5-63 to Figure 5-82.

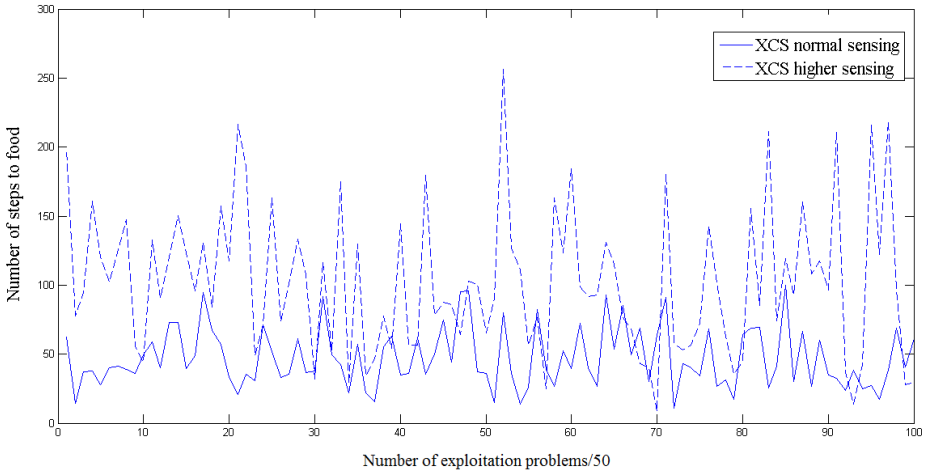


Figure 5-63: The results of learning of XCS animat with normal vision and higher vision abilities in Complex1 environment.

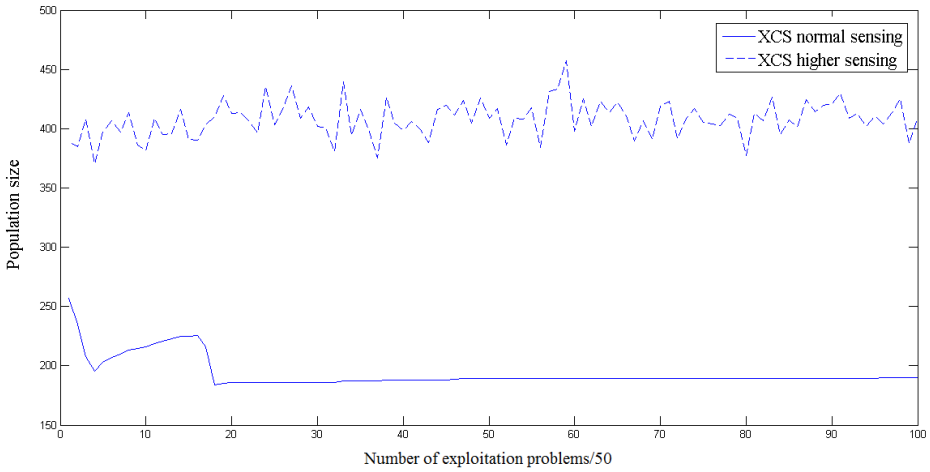


Figure 5-64: The population size of classifiers with normal vision and higher vision abilities in Complex1 environment (XCS).

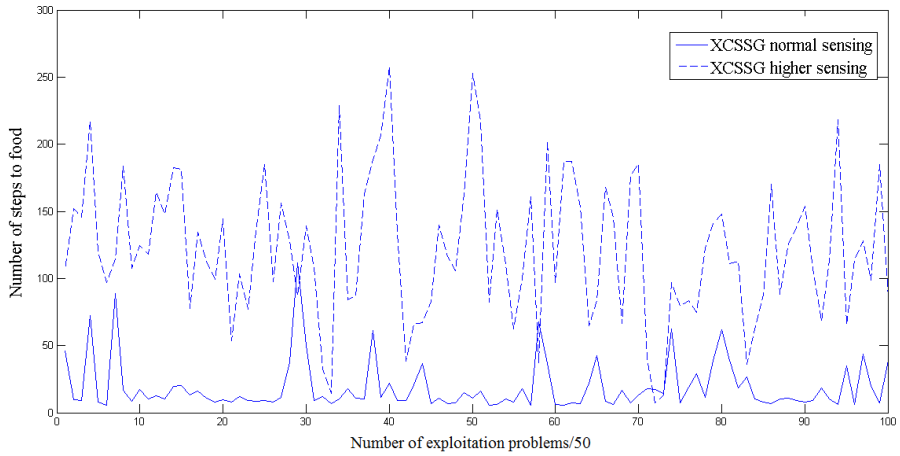


Figure 5-65: The results of learning of XCSSG animat with normal vision and higher vision abilities in Complex1 environment.

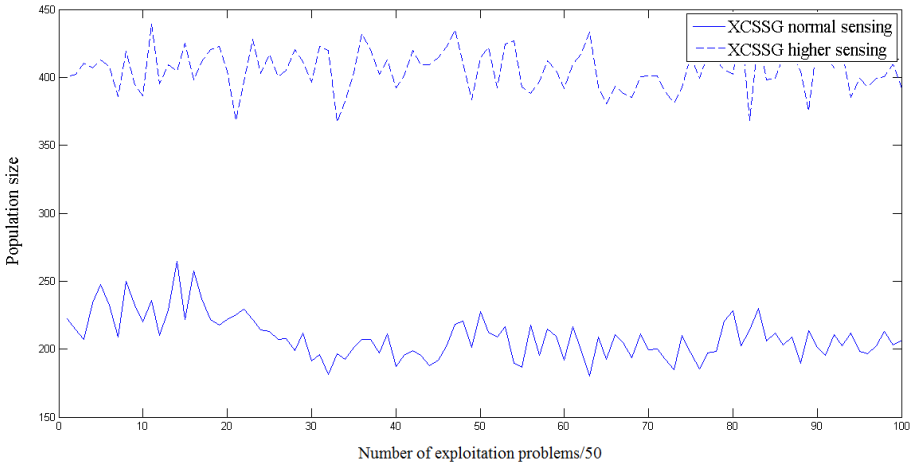


Figure 5-66: The population size of classifiers with normal vision and higher vision abilities in Complex1 environment (XCSSG).

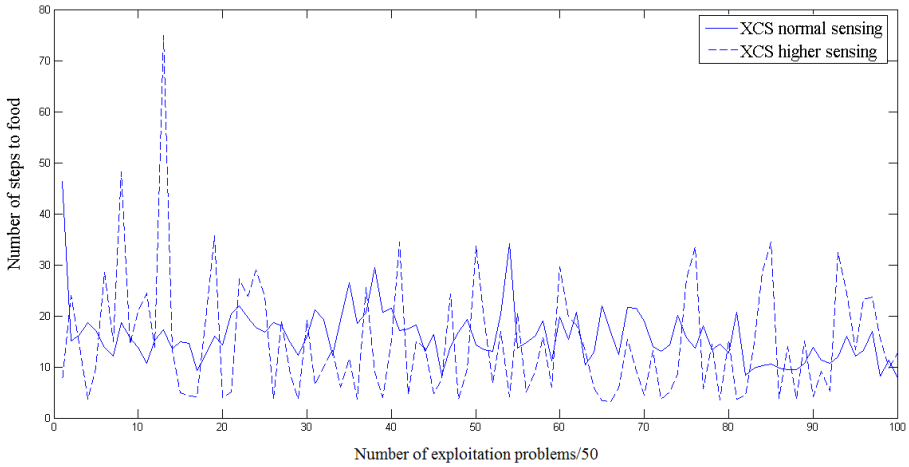


Figure 5-67: The results of learning of XCS animat with normal vision and higher vision abilities in Complex2 environment.

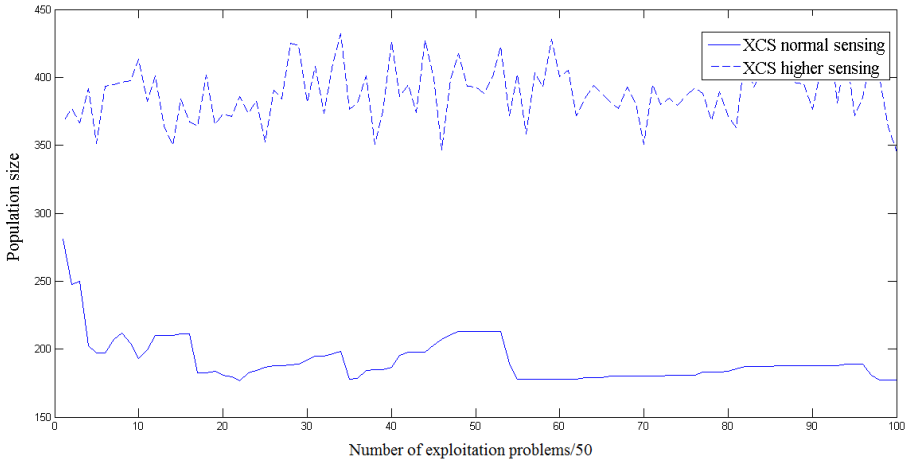


Figure 5-68: The population size of classifiers with normal vision and higher vision abilities in Complex2 environment (XCS).

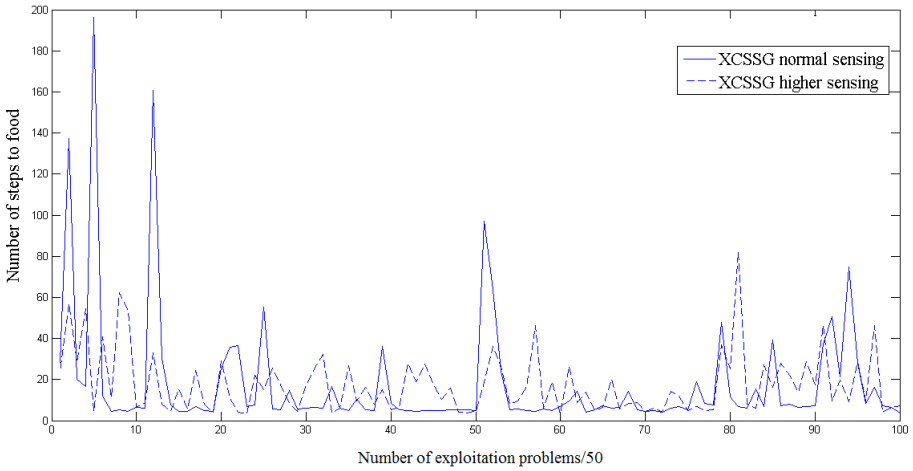


Figure 5-69: The results of learning of XCSSG animat with normal vision and higher vision abilities in Complex2 environment.

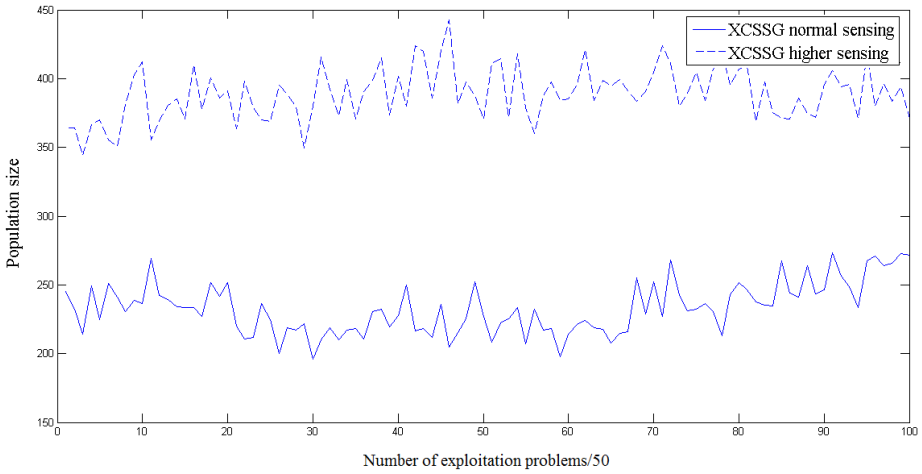


Figure 5-70: The population size of classifiers with normal vision and higher vision abilities in Complex2 environment (XCSSG).

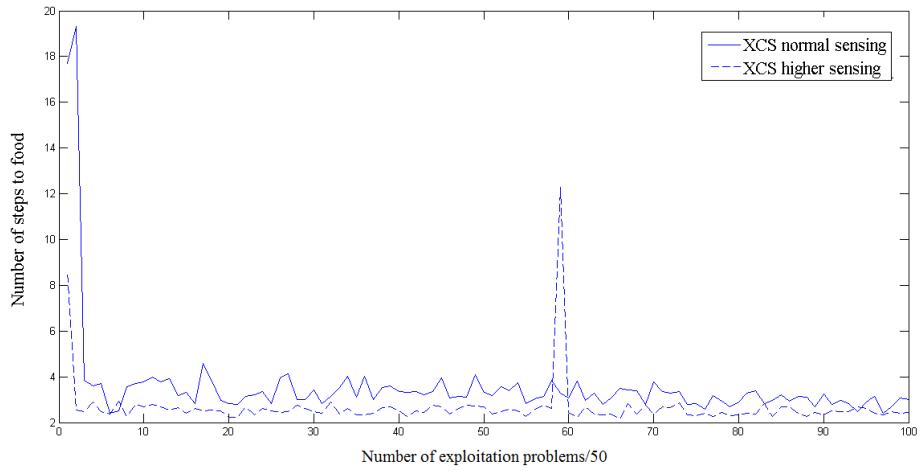


Figure 5-71: The results of learning of XCS animat with normal vision and higher vision abilities in Complex3 environment.

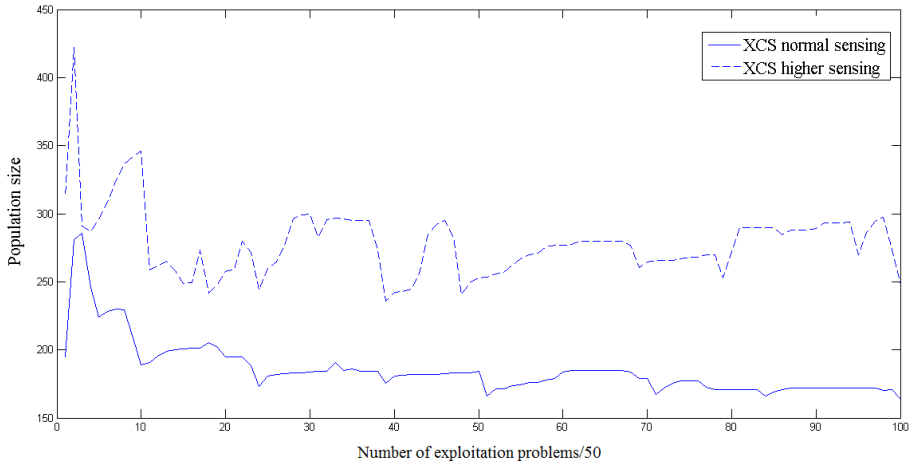


Figure 5-72: The population size of classifiers with normal vision and higher vision abilities in Complex3 environment (XCS).

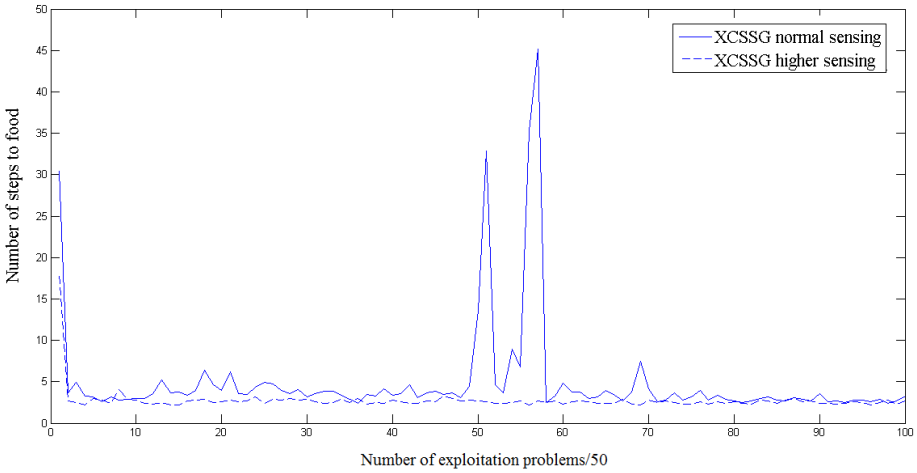


Figure 5-73: The results of learning of XCSSG animat with normal vision and higher vision abilities in Complex3 environment.

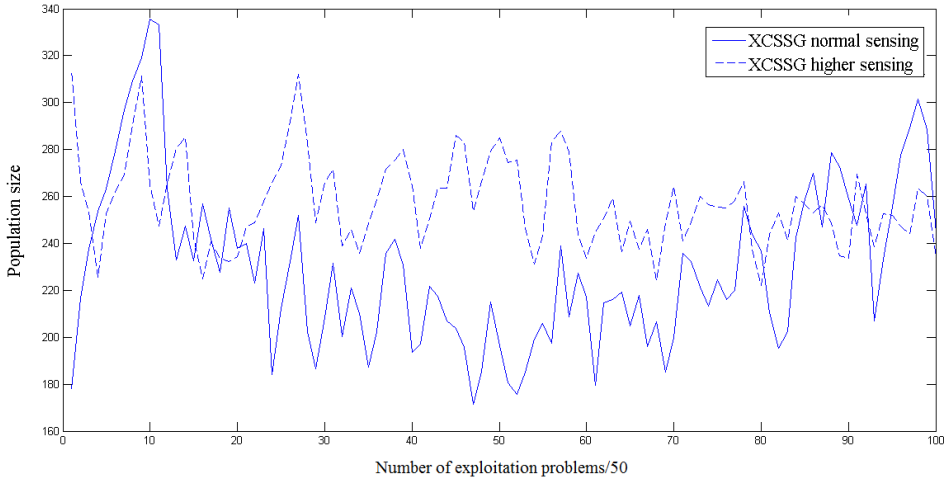


Figure 5-74: The population size of classifiers with normal vision and higher vision abilities in Complex3 environment (XCSSG).

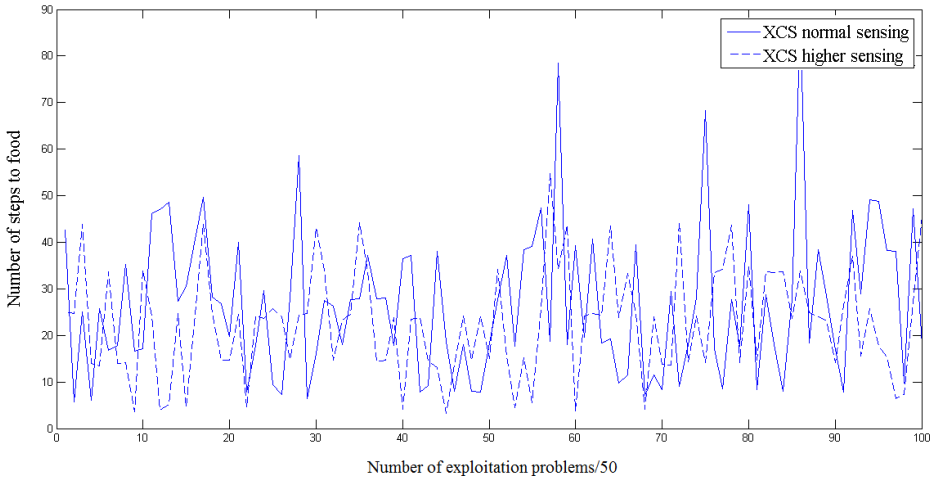


Figure 5-75: The results of learning of XCS animat with normal vision and higher vision abilities in Complex4 environment.

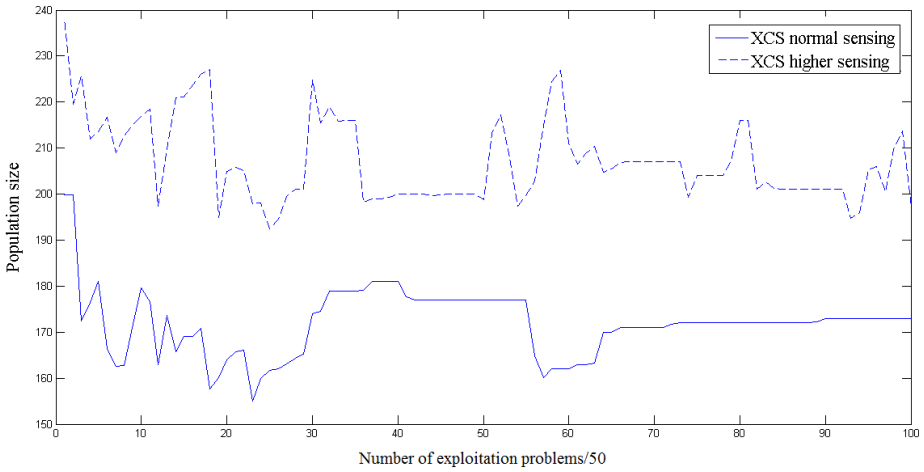


Figure 5-76: The population size of classifiers with normal vision and higher vision abilities in Complex4 environment (XCS).

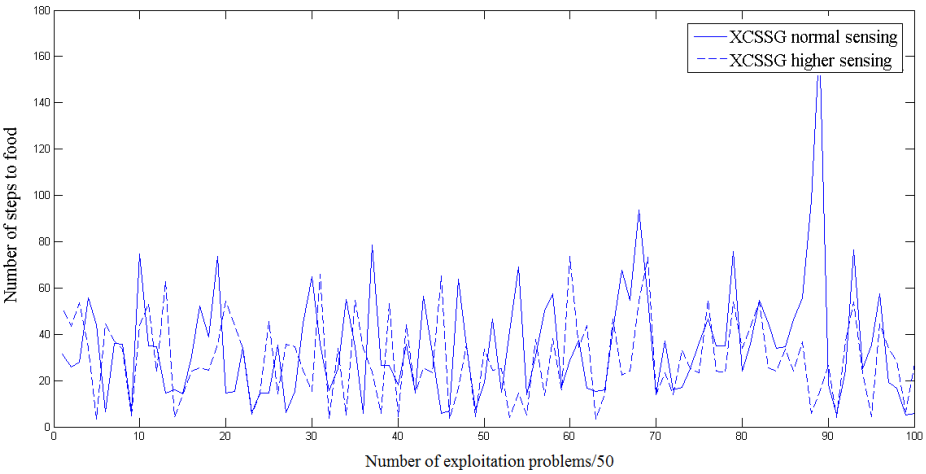


Figure 5-77: The results of learning of XCSSG animat with normal vision and higher vision abilities in Complex4 environment.

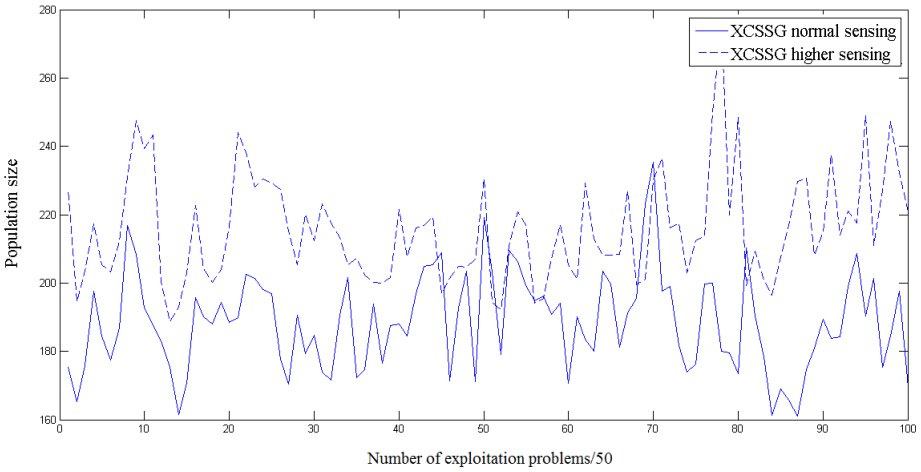


Figure 5-78: The population size of classifiers with normal vision and higher vision abilities in Complex4 environment (XCSSG).

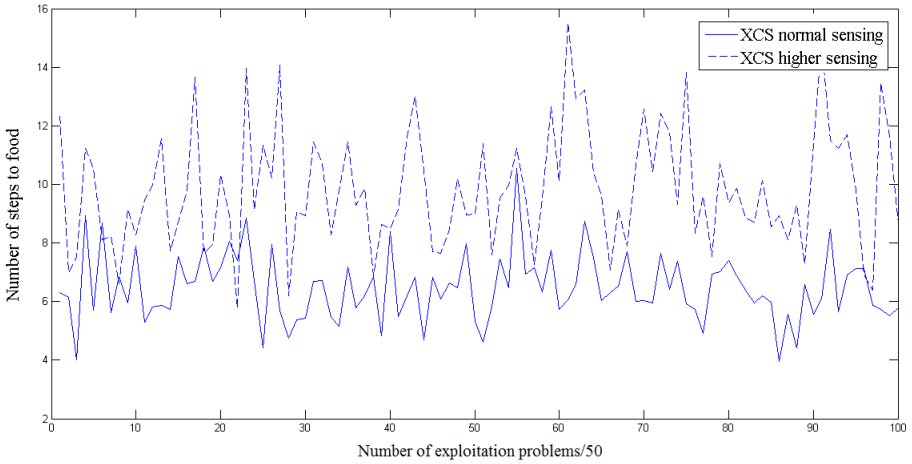


Figure 5-79: The results of learning of XCS animat with normal vision and higher vision abilities in woods101 environment.

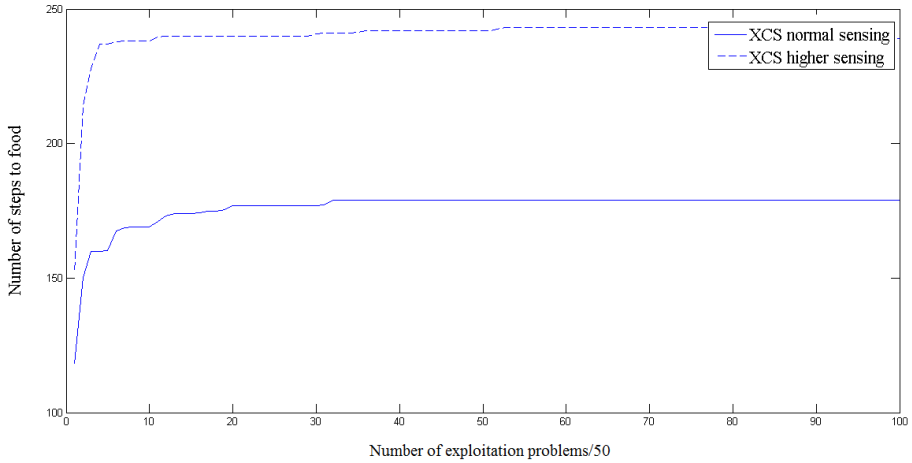


Figure 5-80: The population size of classifiers with normal vision and higher vision abilities in woods101 environment (XCS).

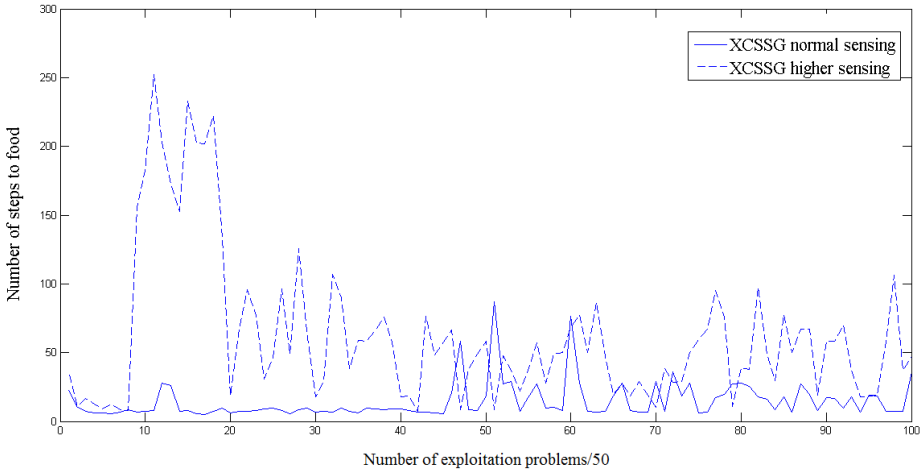


Figure 5-81: The results of learning of XCSSG animat with normal vision and higher vision abilities in woods101 environment.

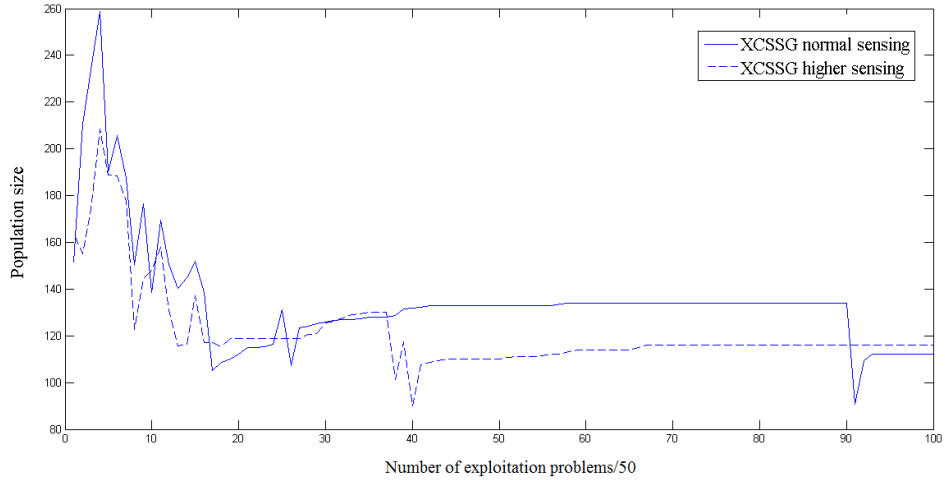


Figure 5-82: The population size of classifiers with normal vision and higher vision abilities in woods101 environment (XCSSG).

The results show that although we expect that adding more sensory information at each step improves the performance of XCS in non-Markovian environments but the results show another thing. Using information of the farther environment at each step makes a non-Markovian environment a Markovian environment but this doesn't help animat to improve its performance and choose the optimal action at each step. The results for XCS and XCSSG at this case are nearly the same. The reason for this behavior is an open problem for this thesis and is work of the future researches. So, we can conclude that for a non-Markovian environment we need a more powerful approach.

5.6 Comparison of mean and variance in different environments

To compare the performances of XCS-family algorithms in different environments the values of means and variances of different learning algorithms (XCS-family) have been calculated and presented in the tables 5.1 to 5.3.

Table 5.1:Comparison of Means and Variances in different generated environments.

	Mean	**Variance**
5MS2DM2:XCS	1.6714	0.0047
5MS2DM2:XCS with subsumption	1.6643	0.0049
5MS2DM2: XCSG	1.6576	0.0045
5MS2DM2:XCSS	1.6780	0.0039
5MS2DM2:XCSSG	1.6698	0.0050
6MS2DM3:XCS	1.5894	0.0103
6MS2DM3:XCS with subsumption	1.7027	0.0167
6MS2DM3:XCSG	1.5888	0.0084
6MS2DM3:XCSS	1.5863	0.0077
6MS2DM3:XCSSG	1.5986	0.0070
7MS2DM6:XCS	2.2333	0.0342
7MS2DM6:XCS with subsumption	2.0549	0.0147
7MS2DM6:XCSG	2.6241	0.0508
7MS2DM6:XCSS	3.1365	30.3620
7MS2DM6:XCSSG	2.0649	0.0180
7nMS2DM6:XCS	2.4014	0.0403
7nMS2DM6:XCS with subsumption	2.2624	0.0225
7nMS2DM6:XCSG	2.1927	0.0252
7nMS2DM6:XCSS	2.2512	0.0431
7nMS2DM6:XCSSG	2.3804	0.6363
7MS2DM8:XCS	2.4194	0.0362
7MS2DM8:XCS subsumption	2.2653	0.0313
7MS2DM8:XCSG	2.6378	0.5881
7MS2DM8:XCSS	2.3671	1.7141
7MS2DM8:XCSSG	2.5610	5.3820

Table 5.2: Comparison of Means and Variances in different traditional environments.

	Mean	**Variance**
Woods1:XCS	1.8900	0.0018
Woods1:XCS with subsumption	1.8174	0.0013
Woods1: XCSG	1.9240	0.0146
Woods1:XCSS	1.6801	0.0020
Woods1:XCSSG	1.8270	0.0079
Woods2:XCS	1.831	0.0015
Woods2:XCS with subsumption	1.8943	0.0024
Woods2:XCSG	1.8285	0.0100
Woods2:XCSS	1.8537	0.0139
Woods2:XCSSG	1.7731	0.0230
Maze5:XCS	10.2880	1.1788
Maze5:XCS with subsumption	415.6876	12817
Maze5:XCSG	16.0424	5.3469
Maze5:XCSS	15.4997	297.8519
Maze5:XCSSG	7.3294	0.8639

Table 5.3: Comparison of Means and Variances in different Complex-family environments.

	Mean	**Variance**
Complex1:XCS normal sensing	47.3918	462.7
Complex1:XCS higher sensing	101.5514	2769.8
Complex1:XCSSG normal sensing	18.9802	364.3
Complex1:XCSSG higher sensing	121.3204	2858.7
Complex2:XCS normal sensing	15.6622	20.6
Complex2:XCS higher sensing	14.6033	133.5
Complex2:XCSSG normal sensing	16.7339	818.8
Complex2:XCSSG higher sensing	17.0504	213.2
Complex3:XCS normal sensing	3.2443	0.2
Complex3:XCS higher sensing	2.6112	1.0
Complex3:XCSSG normal sensing	4.7343	37.9
Complex3:XCSSG higher sensing	2.5931	0.1
Woods101:XCS normal sensing	6.4555	1.3
Woods101:XCS higher sensing	9.8324	4.1
Woods101:XCSSG normal sensing	14.5557	181.4
Woods101:XCSSG higher sensing	63.2855	2918.9

5.7 Conclusion

In this chapter several environments were introduced and for each one the XCS-family algorithms were tested to show the ability of XCS in learning different Markovian and also simple non-Markovian environments. An analysis of generalization was performed based on the change in the number of classifiers in the population set. A comparison is made between the generalization abilities of different algorithms in several environments. To study the ability of animat in changing environment an unstable resource scenario was made in 7MS2DM6 environment to show the change in performance and number of classifiers of each algorithm and their sensitivity to the changes. Problem of interspecific competition were studied based on the competitive Lotka-Volterra equation for competition between XCS and XCSSG. It was shown that in competition XCS and XCSSG are approximately the same and there is no significant

difference in their performance. At the end of the chapter the ability of animat to tackle with some complex non-Markovian environment was tested by observing farther distance cells to give it the ability of converting a non-Markovian environment to Markovian environments. Using XCS and XCSSG to learn this kind of problems didn’t achieve an acceptable performance, didn’t approach to the optimal performance, and opened a question for explaining the reason for this kind of behavior of XCS.

CONCLUSION AND FUTURE WORKS

In Chapter 1 the animat problem, definition, its components, reinforcement learning animat, and Wilson's animat were introduced. It was shown that in RL animat the environment contains objects that each has a reward and animat learns based on the distributed reward. In Chapter 2 the concept of reinforcement learning problem and the methods to solve a reinforcement learning problem were introduced and it was noted that Q-learning is one of the well-known methods to solve a reinforcement learning problem. Learning classifier systems and genetic algorithm that are the main building blocks of solutions for Wilson's animat problem were presented in Chapter 2. In Chapter 3 XCS classifier systems were presented as the main algorithm in this thesis to be used for Wilson's animat problem. XCS is chosen because of its generalization ability (traditional RL algorithms such as Q-learning don't have this ability) and also its performance that is superior to other learning classifier systems. At the end of Chapter 3 different approaches of XCS classifier systems were introduced to show their flexibility to be used in various situations. In Chapter 4 developments to XCS were introduced to remove over-general classifiers and increase the performance of XCS. In Chapter 5 several new environments and scenarios were presented to investigate the ability of XCS-family algorithms for new problems beyond the traditional works.

In this thesis the animat problem was discussed, in different forms, and a specific kind of animat problem studied in different environments. The considered animat problem is called XCS animat problem which is a specific kind of reinforcement learning problem. The XCS animat problem was tested in different environments and the results then were compared to show the strength points and weaknesses of different XCS algorithms. XCS may fail in some environments to converge to optimal solution. Based on the previous works on XCS animat, two improvements on XCS were introduced and based on their combination a new XCS was proposed called XCSSG. The performance of XCSSG was compared with the previous methods in different environments and showed that *specify* operator and gradient descent together can improve the learning of animat. To present the ability of XCS beyond the traditional works based on the literature, several Markovian environments were introduced and XCS-family environments were tested on them and the results of performance and generalization ability were compared. It was shown that XCS can learn simply in a high range of Markovian environments. To give a better insight into the operation of XCS in changing environments, an unstable resource scenario was introduced

and XCS-family algorithms were tested. It was shown that XCSG is the best algorithm to tackle to this kind of problem. A competition platform was developed based on competitive Lotka-Volterra equation to compare the performance of XCS and XCSSG in competition. The results showed that their performances in competition are close and the probability of selecting one of the algorithms remains around 0.5. The ability of animat to observe not only one-step cells but also farther cells may convert a non-Markovian environment to a Markovian environment. This property was used as a basis to test XCS for non-Markovian environments for animats with higher vision abilities. The learning behavior was shown that the performance doesn't converge to the optimal value and using this idea doesn't improve the learning of animat. So, it opened a question for this kind of behavior for future studies. At the end we can conclude that for Markovian environments such as woods1, woods2, 5MS2DM2, 6MS2DM3, 7MS2DM6, and 7MS2DM8 XCS alone can give a good learning performance and animat can learn simply. For Markovian environments such as maze5 that generalization mechanism produces over-general classifiers and decreases the performance, XCSSG should be used instead. For the kind of unstable resource scenario that was introduced in this thesis XCSG is the best algorithm. In addition note that using subsumption mechanism for XCSG, XCSS, and XCSSG doesn't improve the performance and is not recommended. It is useful only for the problems that generalization ability is very important.

From this thesis we learn that there is no algorithm that can solve and work for every kind of problems and for each environment one type of XCS-family algorithm can achieve better performance. So, it proves the expression for the "No Free Lunch Theorem" that there is no learning algorithm that can learn every kind of data sets. From this thesis one can learn about the learning and generalization ability of XCS classifier systems in several 2-D Markovian environments and its weakness in learning 2-D non-Markovian environments. The ability of XCSG in adaptating to a changing environment is another important conclusion of this thesis.

For the future works, finding the reason for this kind of learning behavior for XCS animat with higher vision abilities can open doors to new abilities and behaviors of XCS. Possible changes in the coding of the objects can solve this problem (as a hypothesis). Addition of memory to XCSSG is another development for XCS that can be applied to non-Markovian environments such as woods101, woods 101 $\frac{1}{2}$, and woods102. Modeling of the environments with cellular

automata may lead to the development of XCS-cellular automata for Markovian environments. For example, a three dimensional maze environment where the number of surrounding cells and the number of actions are 26 (cube maze), or a two dimensional polygonal environment can be considered.

REFERENCES

[1] S. W. Wilson, "Classifier Fitness Based on Accuracy," *Evolutionary Computation,* vol. 3, pp. 149-175, 1995.

[2] S. W. Wilson, "KNOWLEDGE GROWTH IN AN ARTIFICIAL ANIMAL," presented at the Proceedings of the 1st International Conference on Genetic Algorithms, 1986.

[3] J. A. Meyer, "Artificial life and the animat approach to artificial intelligence," *Artificial intelligence* pp. 325–354, 1996.

[4] M. A. Bedau, "Artificial life: organization, adaptation and complexity from the bottom up," *Trends in Cognitive Scinces,* vol. 7, pp. 505-512, Nov. 2003.

[5] P. M. W. Todd, S.W.; Somayaji, A.B.; Yanco, H.A., "The blind breeding the blind: Adaptive behavior without looking," presented at the Proceedings of 3rd International Conference on Simulation of Adaptive Behaviour, Brighton, U.K., 1994.

[6] S. Wilson, "The animat path to AI," in *1st International Conf On Simulation Of Adaptive Behavior : From Animals To Animats*, Paris, France, 1991.

[7] F. Krebs and H. Bossel, "Emergent value orientation in self-organization of an animat," *Ecological Modelling,* vol. 96, pp. 143-164, Mar. 1997.

[8] J. A. Meyer, "From natural to artificial life: Biomimetic mechanisms in animat designs," *Robotics and Autonomous Systems,* vol. 22, pp. 3-21, Nov. 10 1997.

[9] J. Kodjabachian and J. A. Meyer, "Evolution and development of control architectures in animats," *Robotics and Autonomous Systems,* vol. 16, pp. 161-182, Dec. 1995.

[10] A. Guillot, Meyer, J.A., "Synthetic Animals in Synthetic Worlds," *Springer Verlag*, pp. 144 -153, 1994.

[11] J. A. Meyer, "The animat approach: Simulation of adaptive behavior in animals and robots," presented at the NPI, 1998.

[12] M. Dorigo, "Editorial Introduction to the Special Issue on Learning Autonomous Robots," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 26, pp. 361–364, 1996.

[13] J. M. Watts, "Animats: Computer-simulated animals in behavioral research," *Journal of Animal Science,* vol. 76, pp. 2596-2604, Oct 1998.

[14] J. Kodjabachian and J. A. Meyer, "Evolution and development of modular control architectures for 1D locomotion in six-legged animats," *Connection Science,* vol. 10, pp. 211-237, Dec 1998.

[15] A. Guillot and J.-A. Meyer, "The animat contribution to cognitive systems research," *Cognitive Systems Research,* vol. 2, pp. 157-165, 2001.

[16] J. M. Hohendorff and J. S. Rosenthal, "An Introduction to Markov Chain Monte Carlo " University of Toronto2005.

[17] W. C. j. Dayan P., "Reinforcement Learning," in *Encyclopedia of Cognitive Science*, ed.

[18] R. S. Sutton, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts. London, England: A Bradford Book, 2005.

[19] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. thesis, Cambridge University, 1989.

[20] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning,* vol. 8, pp. 279-292, May 1992.

[21] R. J. Urbanowicz and J. H. Moore, "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap," *Journal of Artificial Evolution and Applications,* vol. 2009, pp. 1-25, 2009.

[22] L. B. Richard J. Preen, "Discrete and Fuzzy Dynamical Genetic Programming in the XCSF Learning Classifier System," 2012.

[23] R. A. Richards, "Zeroth-order shape optimization utilizing a learning classifier system," Doctoral Dissertation, Stanford University, Stanford, CA, U.S.A., 1996.

[24] S. W. Wilson, "Generalization in the XCS classifier system," presented at the Proceedings of Genetic Programming Conference (GP-98), Maidson, WI, USA, 1998.

[25] S. W. Wilson, "ZCS: A Zeroth Level Classifier System," *Evolutionary Computation,* vol. 2, pp. 1-18, Spr 1994.

[26] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "Gradient Descent Methods in Learning Classifier Systems: Improving XCS Performance in Multistep Problems," *IEEE Transactions on Evolutionary Computation,* vol. 9, pp. 452-473, 2005.

[27] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* vol. 6, pp. 144-153, 2002.

[28] M. V. Butz, Kovacs, T., Lanzi, Pier Luca, Wilson, Stewart W., "How XCS Evolves Accurate Classifiers," 2001.

[29] P. L. Lanzi, "A Model of the Environment to Avoid Local Learning," 1997.

[30] P. L. Lanzi, "A Study of the Generalization Capabilities of XCS," in *Proceedings of the Seventh International Conference on Genetic Algorithm*, 1997.

[31] L. Bull, & O'Hara, T., "Accuracy-based neuro and neuro-fuzzy classifier systems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 905-911.

[32] L. Bull, and Toby O'Hara, "An Accuracy-based Neural Classifier System," *Technical report, UWE Learning Classifier Systems Group Technical Report-UWELCSG01-008,* 2001.

[33] T. O'Hara and L. Bull, "Backpropagation in Accuracy-Based Neural Learning Classifier Systems," vol. 4399, pp. 25-39, 2007.

[34] T. O'Hara and L. Bull, "Building anticipations in an accuracy-based learning classifier system by use of an artificial neural network," *2005 Ieee Congress on Evolutionary Computation, Vols 1-3, Proceedings,* vol. 3, pp. 2046-2052, 2005.

[35] T. O'Hara and L. Bull, "A memetic accuracy-based neural learning classifier system," *2005 Ieee Congress on Evolutionary Computation, Vols 1-3, Proceedings*, vol. 3, pp. 2040-2045, 2005.

[36] J. Casillas, B. Carse, and L. Bull, "Fuzzy-XCS: A Michigan Genetic Fuzzy System," *IEEE Transactions on Fuzzy Systems*, vol. 15, pp. 536-550, 2007.

[37] J. Casillas, Carse, B., Bull, L., & Carse, B., "Fuzzy XCS: an accuracy-based fuzzy classifier system," presented at the Proceedings of the XII Congreso Espanol sobre Tecnologia y Logica Fuzzy (ESTYLF 2004), 2004.

[38] A. Hamzeh and A. Rahmani, "A fuzzy system to control exploration rate in XCS," *Learning Classifier Systems*, vol. 4399, pp. 115-127, 2007.

[39] A. R. Hamzeh, A ; Parsa, N "Intelligent exploration method to adapt exploration rate in XCS, based on adaptive fuzzy genetic algorithm," presented at the IEEE Conference on Cybernetics and Intelligent Systems, Bangkok, THAILAND, 2006.

[40] S. W. Wilson, "Function Approximation with a Classifier System," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO01)*, 2001, pp. 974–981.

[41] G. Howard, L. Bull, and P.-L. Lanzi, "A spiking neural representation for XCSF," pp. 1-8, 2010.

[42] G. D. B. Howard, Larry ; Lanzi, Pier-Luca "Self-adaptive constructivism in neural XCS and XCSF," in *10th Annual Genetic and Evolutionary Computation Conference, GECCO 2008*, Atlanta, GA, United states, 2008, pp. 1389-1396.

[43] G. D. Howard, L. Bull, and P.-L. Lanzi, "Towards continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF," p. 1219, 2009.

[44] A. Vasilyev, "Autonomous Agent Control Using Connectionist XCS Classifier System," *Transport and Telecommunication*, vol. 3, pp. 56–63, 2002.

[45] S. W. Wilson, "Classifiers that approximate functions," *Natural Computing*, vol. 1, pp. 211-234, 2002.

[46] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Generalization in the XCSF classifier system: analysis, improvement, and extension," *Evol Comput*, vol. 15, pp. 133-68, Summer 2007.

[47] S. W. Wilson, "Classifier conditions using gene expression programming " presented at the 11th International Workshops on Learning Classifier Systems, WLCS 2007, London, United kingdom, 2008.

[48] P. L. Lanzi, D ; Wilson, SW ; Goldberg, DE, "Prediction update algorithms for XCSF: RLS, Kalman filter, and gain adaptation," in *8th Annual Genetic and Evolutionary Computation Conference*, Seattle, WA, 2006, pp. 1505-1512.

[49] P. L. Lanzi, D ; Wilson, SW ; Goldberg, DE, "Extending XCSF beyond linear approximation," presented at the Genetic and Evolutionary Computation Conference, Washington, DC, 2005.

[50] P. L. Lanzi, D ; Wilson, SW ; Goldberg, DE, "XCS with computed prediction for the learning of Boolean Functions," in *IEEE Congress on Evolutionary Computation*, Edinburgh, SCOTLAND, 2005.

[51] S. W. Wilson, "Classifier systems for continuous payoff environments," presented at the Genetic and Evolutionary Computation - GECCO 2004. Genetic and Evolutionary Computation Conference, Seattle, WA, USA, 2004.

[52] S. W. Wilson, "Three architectures for continuous action," *Learning Classifier Systems*, vol. 4399, pp. 239-257, 2007.

[53] T. S. Tran, C ; Duthen, Y ; Nguyen, DT, "XCSF with Computed Continuous Action," in *Annual Conference of Genetic and Evolutionary Computation Conference*, London, ENGLAND, 2007, pp. 1861-1868

[54] P. L. Lanzi, D ; Wilson, SW ; Goldberg, DE, "XCS with computed prediction in continuous multistep environments," in *IEEE Congress on Evolutionary Computation*, Edinburgh, SCOTLAND, 2005, pp. 2032-2039.

[55] P. L. L. Lanzi, D.; Wilson, S.W.; Goldberg, D.E., "XCS with computed prediction in continuous multistep environments," in *The 2005 IEEE Congress on Evolutionary Computation*, Edinburgh, Scotland, UK, 2005.

[56] D. Loiacono and P. L. Lanzi, "Computed Prediction in Binary Multistep Problems," *2008 Ieee Congress on Evolutionary Computation, Vols 1-8*, pp. 3350-3357, 2008.

[57] T. Kovacs, "A Comparison of Strength and Accuracy-Based Fitness in Learnin Classifier Systems," School of Computer Science, University of Birmingham, Birmingham, U.K., 2002.

[58] T. Kovacs, "XCS's Strength-Based Twin: Part I," vol. 2661, pp. 61-80, 2003.

[59] T. Kovacs, "Two views of classifier systems," presented at the 4th International Workshop on Learning Classifier Systems, SAN FRANCISCO, CALIFORNIA, 2002.

[60] T. Kovacs, "Rule Fitness and Pathology in Learning Classifier Systems," *Evolutionary Computation*, vol. 12, pp. 99-135, 2004.

[61] 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*, vol. 8, pp. 28-46, Feb 2004.

[62] P. L. Lanzi, "An Analysis of Generalization in the XCS Classifier System," *Evolutionary Computation*, vol. 7, pp. 125-149, Sum 1999.

[63] M. B. Studley, Larry, "X-TCS: Accuracy-based Learning Classifier System robotics," in *2005 IEEE Congress on Evolutionary Computation, IEEE CEC 2005*, Edinburgh, Scotland, United kingdom, 2005.

[64] R. C. V. Moioli, P.A.; Von Zuben, F.J., "Analysing learning classifier systems in reactive and non-reactive robotic tasks," presented at the Learning Classifier Systems. 10th International Workshop, IWLCS 2006 and 11th International Workshop, IWLCS 2007, Berlin, Germany, 2006.

[65] A. H. Webb, Emma; Ross, Peter; Lawson, Alistair "Controlling a simulated khepera with an XCS classifier system with memory," presented at the 7th European Conference, ECAL 2003, Dortmund, Germany, 2003.

[66] C. Lode, U. Richter, and H. Schmeck, "Adaption of XCS to multi-learner predator/prey scenarios," p. 1015, 2010.

[67] M. Gershoff and S. Schulenburg, "Collective behavior based hierarchical XCS," p. 2695, 2007.

[68] M. V. Butz, K. Sastry, and D. E. Goldberg, "Strong, Stable, and Reliable Fitness Pressure in XCS due to Tournament Selection," *Genetic Programming and Evolvable Machines,* vol. 6, pp. 53-77, 2005.

[69] M. Studley and L. Bull, "Using the XCS classifier system for multi-objective reinforcement learning problems," *Artif Life,* vol. 13, pp. 69-86, Winter 2007.

[70] A. Tomlinson and L. Bull, "An accuracy based corporate classifier system," *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* vol. 6, pp. 200-215, 2002.

[71] D. M. Aliprandi, Alex; Matteucci, Matteo; Bonarini, Andrea, "A Bayesian approach to learning classifier systems in uncertain environments," in *8th Annual Genetic and Evolutionary Computation Conference 2006*, Seattle, WA, U.S.A., 2006, pp. 1537-1544.

[72] P. C. Lanzi, M, "An extension to the XCS classifier system for stochastic environments," presented at the Genetic and Evolutionary Computation Conference (GECCO-99) at the 8th International Conference on Genetic Algorithms/4th Annual Genetic Programming Conference, Orlando, FL, 1999.

[73] S. W. Wilson, "Mining oblique data with XCS," presented at the Advances in Learning Classifier Systems. Third International Workshop, IWLCS 2000., Paris, France, 2001.

[74] P. L. Lanzi, "Adding memory to XCS," *1998 Ieee International Conference on Evolutionary Computation - Proceedings,* pp. 609-614, 1998.

[75] P. L. Lanzi, "Solving Problems in Partially Observable Environments with Classifier Systems (Experiments on Adding Memory to XCS)," 1997.

[76] P. L. Lanzi, "An analysis of the memory mechanism of XCSM," in *Proceedings of Genetic Programming Conference (GP-98)*, Maidson, WI, U.S.A., 1998, pp. 643-51.

[77] P. L. Lanzi and S. W. Wilson, "Toward optimal classifier system performance in non-Markov environments," *Evol Comput,* vol. 8, pp. 393-418, Winter 2000.

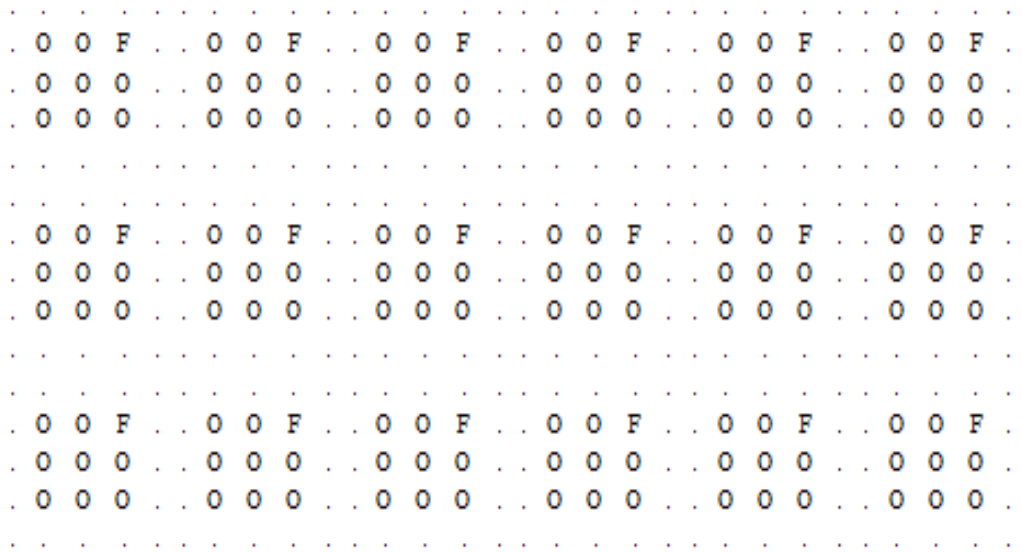
[78] M. V. G. Butz, D.E.; Lanzi, P.L., "Gradient-based learning updates improve XCS performance in multistep problems," in *Genetic and Evolutionary Computation - GECCO 2004. Genetic and Evolutionary Computation Conference*, Seattle, WA, USA, 2004, pp. 751-62.

[79] P. L. Lanzi, M. V. Butz, and D. E. Goldberg, "Empirical Analysis of Generalization and Learning in XCS with Gradient Descent," *Gecco 2007: Genetic and Evolutionary Computation Conference, Vol 1 and 2,* pp. 1814-1821, 2007.

[80] L. G. M. Beals, S. Harrell. (1999). *Interspecific Competition: Lotka-Volterra*. Available: http://www.tiem.utk.edu/~gross/bioed/bealsmodules/competition.html

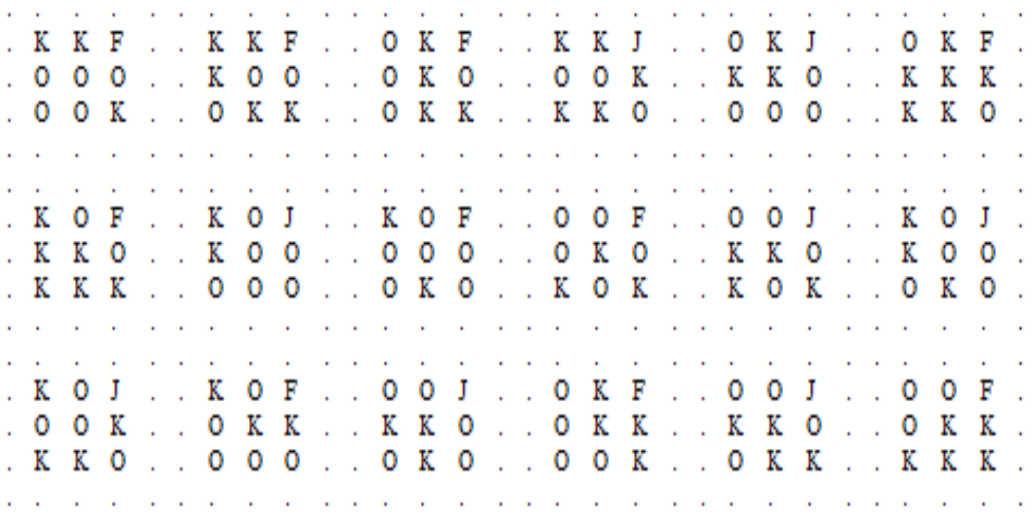
Appendix 1 – Well-known 2-D Environments

Woods1



The environment Woods1

Woods2



The environment Woods2

Maze5

```
O O O O O O O O O
O . . . . . . F O
O . . O . O O . O
O . O . . . . . O
O . . . O O . . O
O . O . O . . O O
O . O . . O . . O
O . . . . . O . O
O O O O O O O O O
```

Maze5 environment

Maze6

```
O O O O O O O O O
O . . . . . O F O
O . . O . O O . O
O . O . . . . . O
O . . . O O . . O
O . O . O . . O O
O . O . . . . . O
O . . . . . O . O
O O O O O O O O O
```

Maze6 environment

Woods14

```
O  O  O  O  O  O  O  O  O  O  O  O  O
O  O  .  .  .  O  O  O  O  .  O  O  .
O  .  O  O  O  .  O  O  .  O  .  O  .
O  .  O  O  O  .  O  .  O  O  O  .  O
O  F  O  O  O  .  O  O  .  O  O  O  O
O  O  O  O  O  O  .  .  O  O  O  O  O
```

Woods14 environment

Maze4

```
O  O  O  O  O  O  O  O
O  .  .  O  .  .  F  O
O  O  .  .  O  .  .  O
O  O  .  O  .  .  O  O
O  .  .  .  .  .  .  O
O  O  .  O  .  .  .  O
O  .  .  .  .  O  .  O
O  O  O  O  O  O  O  O
```

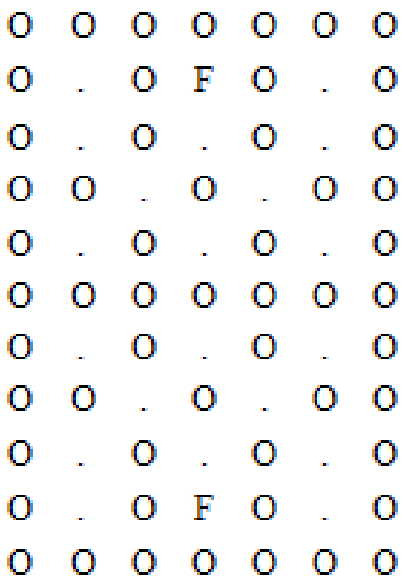
Maze4 environment

Woods101

```
O  O  O  O  O  O  O
O  .  .  .  .  .  O
O  .  O  .  O  .  O
O  .  O  F  O  .  O
O  O  O  O  O  O  O
```

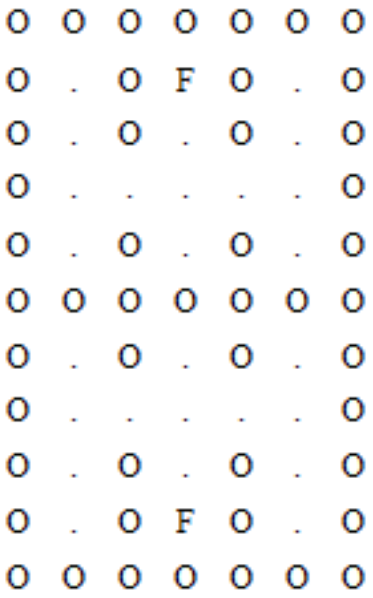
Woods101 environment

Woods101$\frac{1}{2}$



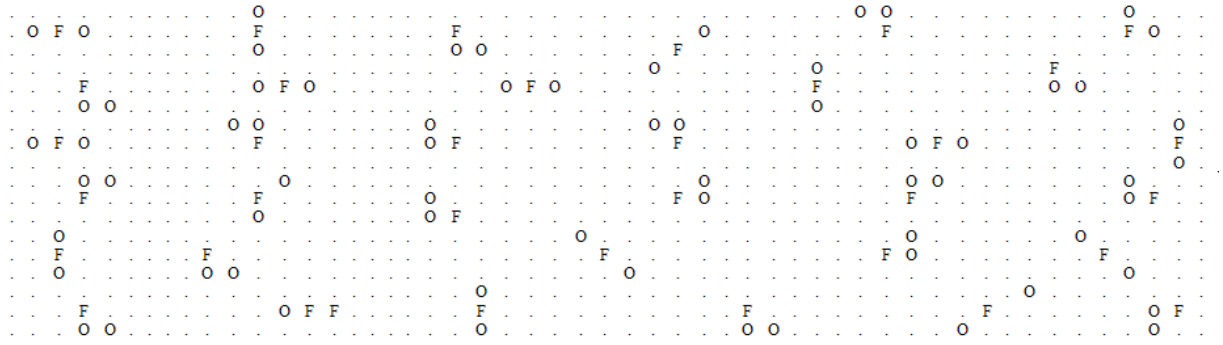
Woods101$\frac{1}{2}$ environment

Woods102



Woods102 environment

Woods7



Woods7 environment