

INSTITUTO TECNOLOGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

CAMPUS MONTERREY

SCHOOL OF ENGINEERING

DIVISION OF MECHATRONICS AND INFORMATION
TECHNOLOGIES

GRADUATE PROGRAMS



DOCTOR OF PHILOSOPHY

IN

INFORMATION TECHNOLOGIES AND
COMMUNICATIONS

MAJOR IN INTELLIGENT SYSTEMS

QFCS: A FUZZY LCS IN CONTINUOUS MULTI-STEP
ENVIRONMENTS WITH CONTINUOUS VECTOR ACTIONS

BY

JOSE ABDON RAMIREZ RUIZ

MAY 2009

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

SCHOOL OF ENGINEERING
DIVISION OF MECHATRONICS AND INFORMATION
TECHNOLOGIES
GRADUATE PROGRAMS



DOCTOR OF PHILOSOPHY
in
INFORMATION TECHNOLOGIES AND COMMUNICATIONS
MAJOR IN INTELLIGENT SYSTEMS
**QFCS: A Fuzzy LCS in Continuous Multi-Step Environments
with Continuous Vector Actions**

By

José Abdón Ramírez Ruiz

MAY 2009

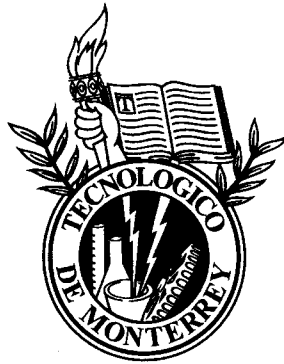
QFCS: A Fuzzy LCS in Continuous Multi-Step Environments with Continuous Vector Actions

A dissertation presented by

José Abdón Ramírez Ruiz

Submitted to the
Graduate Programs in Mechatronics and Information Technologies
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Information Technologies and Communications
Major in Intelligent Systems



Dissertation Committee:

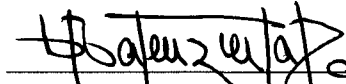
Dr. Manuel Valenzuela Rendón - ITESM, Monterrey Campus
Dr. Hugo Terashima Marín - ITESM, Monterrey Campus
Dr. Santiago Enrique Conant Pablos - ITESM, Monterrey Campus
Dr. Edgar Emmanuel Vallejo C. - ITESM, Edo. de México Campus
Dr. José Torres Jiménez - CINVESTAV-Tamaulipas

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
May 2009

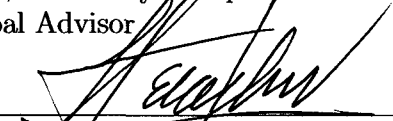
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey Campus

Division of Mechatronics and Information Technologies
Graduate Program

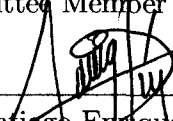
The committee members, hereby, certify that have read the dissertation presented by José Abdón Ramírez Ruiz and that it is fully adequate in scope and quality as a partial requirement for the degree of **Doctor of Philosophy in Information Technologies and Communications**, with a major in **Intelligent Systems**.



Dr. Manuel Valenzuela Rendón
ITESM, Monterrey Campus
Principal Advisor



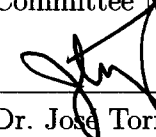
Dr. Hugo Terashima Marín
ITESM, Monterrey Campus
Committee Member



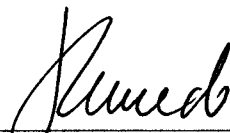
Dr. Santiago Enrique Conant Pablos
ITESM, Monterrey Campus
Committee Member



Dr. Edgar Emmanuél Vallejo C.
ITESM, Edo. de México Campus
Committee Member



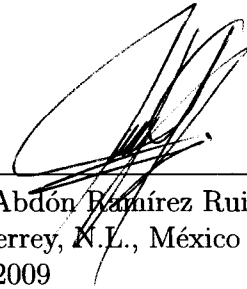
Dr. José Torres Jiménez
CINVESTAV-Tamaulipas
Committee Member



Dr. Joaquín Acevedo M.
Director of Research and Graduate Programs
School of Engineering

Copyright Declaration

I, hereby, declare that I wrote this dissertation entirely by myself and, that, it exclusively describes my own research.



José Abdón Ramírez Ruiz
Monterrey, N.L., México
May 2009

©2009 by José Abdón Ramírez Ruiz
All Rights Reserved

Dedication

To my *family*,
And to that great *dream* that brought me up here.

Acknowledgements

First of all I would like to thank my family for their constant support and encouragement. I would like to express my sincere thanks to my advisor who has provided immense help with preparation of this work. I am very grateful to the Computación Evolutiva Chair for providing a highly stimulating research environment and for giving me the freedom to develop my ideas. And finally I would like to express my deepest gratitude to all those I did not mention who have been side by side with me, along the long, but also short, hours at night.

QFCS: A Fuzzy LCS in Continuous Multi-Step Environments with Continuous Vector Actions

by

José Abdón Ramírez Ruiz

Abstract

This document presents a doctoral dissertation which is a requirement for the Ph.D. degree in Information Technologies and Communications from Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Campus Monterrey, major in Intelligent Systems in the field of Learning Classifier Systems (LCS). The dissertation introduces a new LCS, called QFCS, that is able to deal with problems defined over continuous variables. These problems are important because real life is modeled in that way.

LCSs are systems with a set of rules that compete and that can learn from and adapt to the environment. These properties are very desirable in intelligent systems because they allow the systems to adjust to subtle details. Traditionally, in Artificial Intelligence, designers have to pre-adjust the parameters. This made the developer not to take into account those subtle details and, consequently, deal with them during experimentation. LCSs make use of reinforcement learning and of evolutionary computing to deal with the proper adjustment of those subtleties. But, LCSs in their beginnings have been designed to solve problems that can be defined in a discrete form. Lately, researchers have tried to extend the approach to deal with problems in the continuum. This task has shown to be far away of being solved. Thus, there have been many approaches to adapt these systems to continuous variables. One of them has been the introduction of Fuzzy Logic to model the continuous environment. In this way, little by little LCSs have been extended to tackle more problems in the continuum, increasing, little by little, their related difficulty.

Some of the problems solved with this approaches are the learning of continuous functions, the frog problem and navigation tasks. Learning of continuous functions is a problem where some continuous input enters to the system and the corresponding output is obtained, but the system does not know what this output is, all the system knows is the amount of reward it receives for each output made. This problem is of one-step since the system has to place an output once. The frog problem consists of a frog that lives with a fly in a line. The frog has to jump once and catch the fly. Since the frog lives in a line, the environment is continuous. The length the frog jumps is also continuous. This is one-step since the frog jumps once. The frog receives a reward at each time it jumps even if it does not trap the fly. Navigation tasks are more complex problems since they are multi-step. This means the system has to act more than one time to reach the goal. In this case, it moves many times to reach another place. The environment can be discrete but it is more complex if it is continuous. The actions are a set of discrete vectors but, in a more complex form, they could be continuous. The reward is given when the system reaches the goal.

The complexity for LCSs with the problems described before is given by the set of continuous actions, because rules of LCSs relate the states of the problem with one action. Thus, to model continuous outputs would required a set of infinite number of rules. This is impossible because rules are countable and the continuum is not.

QFCS introduces fuzzy systems in the rules to model relationships of the form: many states to many actions. This is a novelty in LCSs literature where only single fuzzy rules have been used. QFCS uses a matrix to learn a prediction of the payoff to be obtained per each fuzzy system. This is also a novelty because traditional LCSs use one value to predict the obtained reward. In this way, QFCS was designed following to different approaches, one that has fixed fuzzy sets and the other with unfixed fuzzy sets as inputs to the fuzzy systems. The second approach is a generalization and it is proposed to eliminate the restrictions imposed by the use of fixed fuzzy sets.

This QFCS was tested with the frog problem to compare it with the literature and with five more different problems that introduced different levels of complexity. Three of them were about performing navigation tasks in one and two dimension spaces with obstacles. The last two dealt with an inertial particle. The navigation tasks were in continuous spaces with a set of continuous vector actions. A reward was given in those regions the system had to reach. The particle problems were about moving a particle from one position to another one. These problems were the more complex ones because of the inertia of the particle.

Results showed that QFCS is capable of solving these kind of continuous problems and that there is not too much difference in performance between the two approaches of QFCSs. Liberating the fixed fuzzy sets did not represent an advantage.

List of Figures

1.1	a. Woods1. b. Maze6. c. Woods14	4
1.2	Type of problem.	8
2.1	Membership functions of linguistic values	15
2.2	Cylindrical extensions of $\mu_{X_{13}}(x_1)$ and $\mu_{X_{25}}(x_2)$	16
2.3	Projection of $\mu(x, a)$ on a	19
3.1	Table that represents the Q-values of $Q(x, a)$	24
3.2	Graph that represents the transition function of the perceived environment.	25
3.3	a. Three states (x, a) with their transitions. b. One sequence of actions of an agent.	25
3.4	Updating of Q-values on time.	29
4.1	Performance Component of the Holland's LCS	33
4.2	Rule chaining	35
4.3	Performance Component of the XCS	37
4.4	Learning Component of the XCS	38
4.5	Woods1	40
4.6	Performance component of the XCSF	41
4.7	The linear corridor problem and the 2D GridWorld problem	43
4.8	Membership functions of a linguistic variable in Valenzuela's FCS	44
4.9	Diagram of the performance component of the Valenzuela's FCS.	47
4.10	Diagram of the performance component of the Parodi and Bonelli's FCS.	49
4.11	Membership functions of a linguistic variable in Bonarini's FCS	51
4.12	Diagram of the performance component of the Bonarini's FCS.	53
4.13	Corridor problem with a CAT robot.	55
5.1	Diagram of components of the QFCS	58
5.2	a. Input space $x_1 \dots x_n$ with $n = 2$. b. Square regions R_{c_1, \dots, c_n} with $c = 4$. c. Square sub-regions Δ_{d_1, \dots, d_n} with $d = 4$. d. j -th component of the vector field $\vec{a}_i = (a_1^i, \dots, a_m^i)$	59
5.3	Diagram of performance component	60
5.4	a. Meaning of a classifier. b. Performance component. c. Meaning of the prediction vectors	61
5.5	a. Component a_1^i of classifiers $Cl_i \in [M]$. b. Component a_2^i of classifiers $Cl_i \in [M]$. c. Matrix of elements p_{d_1, d_2}^1 of the classifier Cl_1 . d. Matrix of elements p_{d_1, d_2}^2 of the classifier Cl_2	62

5.6	Learning component	64
5.7	a. QFCS at time $(t - 1)$. b. QFCS at time t	64
5.8	a. Input space $x_1 \dots x_n$ with $n = 2$. b. Square region R^l . c. Square sub-regions $\Delta_{d_1, \dots, d_n}^l$ with $d = 4$. d. j -th component of the vector field $\vec{a}_l = (a_1^l, \dots, a_m^l)$	70
5.9	a. Meaning of a classifier. b. Performance component. c. Meaning of the prediction vectors	71
5.10	a. Component a_1^l of classifiers $Cl_l \in [M]$. b. Component a_2^l of classifiers $Cl_l \in [M]$. c. Matrix of elements p_{d_1, d_2}^1 of the classifier Cl_1 . d. Matrix of elements p_{d_1, d_2}^2 of the classifier Cl_2	73
5.11	Learning component	74
5.12	a. QFCS at time $(t - 1)$. b. QFCS at time t	75
5.13	a. Memberships of the fuzzy sets from SFS_l for the input variables x_i . b. Memberships of the fuzzy sets from SFS_l for the output variables a_j	78
5.14	a. Fuzzyfication and cylindrical extension of the input vector \vec{x}_0 . b. Membership function of the implication R_1^l . c. Membership function of the implication R_2^l . d. Membership function of the inference operation with fuzzy rule R_1^l . e. Membership function of the inference operation with fuzzy rule R_2^l . f. Membership function of the union over the results of the inference operations.	80
6.1	a. Reward function $R(x, a)$. b. Optimal solution	84
6.2	a. $World_a^{1D}$. b. Optimal solution of $World_a^{1D}$	86
6.3	a. $World_a^{2D}$. b. Optimal solution of $World_a^{2D}$	87
6.4	a. $World_b^{2D}$. b. Optimal solution of $World_b^{2D}$	88
6.5	a. $Particle_a^{1D}$. b. Optimal solution of $Particle_a^{1D}$	90
6.6	a. $Particle_b^{1D}$. b. Optimal solution of $Particle_b^{1D}$	92
6.7	a. Initial conditions of the particle with mass m	92
7.1	a. Initial State of the QFCS with fixed fuzzy sets in the Frog Problem. b. Initial State of the QFCS with unfixed fuzzy sets in the Frog Problem. c. Initial State of Q-learning in the Frog Problem.	98
7.2	Three instances of the QFCS with fixed fuzzy sets in the Frog Problem.	100
7.3	Three instances of the QFCS with unfixed fuzzy sets in the Frog Problem.	101
7.4	Three instances of the Q-learning in the Frog Problem.	102
7.5	a. Average over 20 runs of the action learned in the Frog Problem. b. Average over 20 runs of the error in the Frog Problem.	103
7.6	a. Initial State of the QFCS with fixed fuzzy sets in $World_a^{1D}$. b. Initial State of the QFCS with unfixed fuzzy sets in $World_a^{1D}$. c. Initial State of Q-learning in $World_a^{1D}$	104
7.7	Three instances of the QFCS with fixed fuzzy sets in $World_a^{1D}$	106
7.8	Three instances of the QFCS with unfixed fuzzy sets in $World_a^{1D}$	107
7.9	Three instances of the Q-learning in $World_a^{1D}$	108

7.10	a. Average over 20 runs of the action learned in World_a^{1D} . b. Average over 20 runs of the number of steps to goal against x_1 in World_a^{1D} . c. Average over 20 runs of the number of steps to goal against trials in World_a^{1D} . . .	109
7.11	a. Initial State of the QFCS with fixed fuzzy sets in World_a^{2D} . b. Initial State of the QFCS with unfixed fuzzy sets in World_a^{2D} . c. Initial State of Q-learning in World_a^{2D}	111
7.12	Three instances of the QFCS with fixed fuzzy sets in World_a^{2D}	111
7.13	Three instances of the QFCS with unfixed fuzzy sets in World_a^{2D}	112
7.14	Three instances of Q-learning in World_a^{2D}	113
7.15	a. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with QFCS with fixed fuzzy sets. b. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with QFCS with unfixed fuzzy sets. c. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with Q-learning. d. Average over 20 runs of the number of steps to goal against trials in World_a^{1D}	114
7.16	a. Initial State of the QFCS with fixed fuzzy sets in World_b^{2D} . b. Initial State of the QFCS with unfixed fuzzy sets in World_b^{2D} . c. Initial State of Q-learning in World_b^{2D}	115
7.17	Three instances of the QFCS with fixed fuzzy sets in World_b^{2D}	116
7.18	Three instances of the QFCS with unfixed fuzzy sets in World_b^{2D}	117
7.19	Three instances of Q-learning in World_b^{2D}	117
7.20	a. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by QFCS with fixed fuzzy sets c. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by Q-learning. d. Average over 20 runs of the number of steps to goal against trials in World_b^{1D}	118
7.21	a. Initial State of the QFCS with fixed fuzzy sets in Particle_a^{1D} . b. Initial State of the QFCS with unfixed fuzzy sets in Particle_a^{1D} . c. Initial State of Q-learning in Particle_a^{1D}	120
7.22	Three instances of the QFCS with fixed fuzzy sets in Particle_a^{1D}	120
7.23	Three instances of the QFCS with unfixed fuzzy sets in Particle_a^{1D}	121
7.24	Three instances of Q-learning in Particle_a^{1D}	122
7.25	a. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with unfixed fuzzy sets. c. Average over 20 runs of the force and number of steps to goal against xv obtained by Q-learning. d. Average over 20 runs of the number of steps to goal with $v = 0$ against x in Particle_a^{1D} . e. Average over 20 runs of the number of steps to goal against trials in Particle_a^{1D}	123
7.26	a. Initial State of the QFCS with fixed fuzzy sets in Particle_b^{1D} . b. Initial State of the QFCS with unfixed fuzzy sets in Particle_b^{1D} . c. Initial State of Q-learning in Particle_b^{1D}	123

7.27	Three instances of the QFCS with fixed fuzzy sets in Particle _b ^{1D}	124
7.28	Three instances of the QFCS with unfixed fuzzy sets in Particle _b ^{1D}	125
7.29	Three instances of Q-learning in Particle _b ^{1D}	126
7.30	a. Average over 20 runs of the force and number of steps to goal against <i>xv</i> obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the force and number of steps to goal against <i>xv</i> obtained by QFCS with fixed fuzzy sets. c. Average over 20 runs of the force and number of steps to goal against <i>xv</i> obtained by Q-learning. d. Average over 20 runs of the number of steps to goal with $v = 0$ against <i>x</i> in Particle _b ^{1D} . e. Average over 20 runs of the number of steps to goal against trials in Particle _b ^{1D}	127
8.1	Curves generated by one classifier in combination with all of its close classifiers	136

List of Tables

1.1	Classification of the LCSs. D=Discrete, C=Continuous, []=Maximum number used in experiments.	9
5.1	Parameters of QFCS with fixed fuzzy sets.	67
5.2	Spatial complexity of different parts of QFCS with fixed fuzzy set.	67
5.3	Temporal complexity of different parts of one cycle of QFCS with fixed fuzzy set.	67
5.4	Parameters of QFCS with unfixed fuzzy sets.	76
5.5	Spatial complexity of different parts of QFCS with unfixed fuzzy set.	77
5.6	Temporal complexity of different parts of one cycle of QFCS with unfixed fuzzy set.	77
7.1	General parameters in all of the experiments.	96
7.2	Particular parameters in the Frog Problem.	97
7.3	Particular parameters in World _a ^{1D}	104
7.4	Particular parameters in World _a ^{2D} and World _b ^{2D}	110
7.5	Particular parameters in Particle _a ^{1D}	119
7.6	Particular parameters in Particle _b ^{1D}	125

Contents

Abstract	ix
List of Figures	xiv
List of Tables	xv
Contents	xvii
1 Introduction	1
1.1 Motivation	2
1.2 Related Works	6
1.3 Problem Statement and Context	7
1.4 Research Questions	8
1.5 Solution Overview	9
1.6 Main Contributions	11
1.7 Dissertation Organization	12
2 Fuzzy Logic	13
2.1 Fuzzy Rules	14
2.2 Membership Functions	14
2.3 Fuzzy Operations	16
2.4 Fuzzification and Defuzzification	21
2.5 General Discussion	22
2.6 Summary	22
3 Q-Learning	23
3.1 The Q-function	24
3.2 Reward Function	25
3.3 Expected Reward	26
3.4 Learning of the Q-function	27
3.5 Exploitation and Exploration	28
3.6 General Discussion	29
3.7 Summary	30

4	Learning Classifier Systems	31
4.1	Hollands LCS (The first LCS)	32
4.2	LCSs with Fitness Based on Accuracy	35
4.2.1	XCS	35
4.2.2	XCSF	39
4.3	Fuzzy LCSs	42
4.3.1	Valenzuela's FCS	43
4.3.2	Parodi and Bonelli's FCS	48
4.3.3	Bonarini's FCS	50
4.4	General Discussion	54
4.5	Summary	56
5	Proposed Solution Model	57
5.1	QFCS with Fixed Fuzzy Sets	57
5.2	QFCS with Unfixed Fuzzy Sets	68
5.3	Small Fuzzy Systems (SFS _{<i>l</i>})	78
5.4	Implementation of Classifiers (Cl_i)	79
5.5	General Discussion	81
5.6	Summary	82
6	Test Problems	83
6.1	The Frog Problem	83
6.2	The n -Environment Problem	85
6.2.1	The 1-Environment Problem: World _{<i>a</i>} ^{1D}	86
6.2.2	The 2-Environment Problem: World _{<i>a</i>} ^{2D}	87
6.2.3	The 2-Environment Problem: World _{<i>b</i>} ^{2D}	88
6.2.4	The 2-Environment Problem: Particle _{<i>a</i>} ^{1D}	89
6.2.5	The 2-Environment Problem: Particle _{<i>b</i>} ^{1D}	91
6.3	Dynamics of an Inertial Particle	91
6.4	General Discussion	93
6.5	Summary	94
7	Experiments and Results	95
7.1	Parameters and Experiment Structure	95
7.2	The Frog Problem	97
7.3	The 1-Environment Problem: World _{<i>a</i>} ^{1D}	103
7.4	The 2-Environment Problem: World _{<i>a</i>} ^{2D}	109
7.5	The 2-Environment Problem: World _{<i>b</i>} ^{2D}	113
7.6	The 2-Environment Problem: Particle _{<i>a</i>} ^{1D}	119
7.7	The 2-Environment Problem: Particle _{<i>b</i>} ^{1D}	122
7.8	General Discussion	128
7.9	QFCS against other approaches	129
7.10	Summary	130

8 Conclusions	133
8.1 Summary	133
8.2 The Main Essence of QFCS	134
8.3 Implications	135
8.4 Future Work	135
Bibliography	141

Chapter 1

Introduction

Artificial Intelligence (AI) is a huge research area. It deals with the problem of giving intelligence to machines. This has made researchers propose many different approaches to tackle many different problems that people can solve. Traditional AI systems were rigid because they were designed and adjusted a priori by the designer. Then, it was easy to overlook subtle details that were needed for the optimal performance of the system. In this way, Machine learning and evolutionary approaches have tried to make systems more capable of changing in its internal parameters and in its structure to adapt to the problem.

Evolutionary approaches are techniques that simulate the processes found in the evolution theory. Meaning that, there is a population called individuals. These individuals form part of some generation. Generations are used to produce new generations selecting, crossing and mutating the strongest individuals. Evolutionary approaches have been used mainly for optimization. In the evolutionary approaches each individual is a possible solution of the problem to solve and when the evolutionary process finishes the strongest individual is the best solution. Therefore the individual is adapted in its structure. There are many different evolutionary algorithms but one widely used is the Genetic Algorithm (GA) introduced by Holland [12, 11, 10].

On the other hand, machine learning allows systems to adapt to changes in the problem so the system designer does not need to foresee and provide solutions for all possible situations. There are mainly three branches in machine learning: Supervised, Unsupervised and Reinforcement Learning (RL) [2, 1, 27]. The last one is a technique based on punishment and reward, in other words, the way the system learns is similar to animals being trained, where punishment is represented by pain and reward by instant gratification. Feeling pain makes animals avoid bad behavior, whereas gratification stimulates them to keep their good behavior. In reinforcement learning, reward can be a negative or positive number representing pain and pleasure respectively. The system maintains a set of parameters that contains a prediction of the received accumulative reward, so the system follows the behavior that predict to obtain the most reward possible. For reinforcement learning, the problem has to be coded into states with a set of actions and with a transition function that defines how the problem moves between two different states due to a particular action. One of the algorithms in reinforcement learning that has had a lot of success is the Q-learning algorithm [1, 27]. This algorithm does

not need the transition function. Q-learning was designed to solve discrete problems. That means that the states and the actions of the problem are discrete sets. Therefore, to solve a navigation problem in a real space, which is continuous and with real actions, which are also continuous, the space and the actions have to be highly discretized to be trustworthy. The problem with having a highly discretized world is the need of a lot of memory to save the parameters causing the algorithm to stop performing well.

Continuous problem have been dominated by control theory. Zadeh in 1965 introduced Fuzzy Logic (FL) [37, 32]. FL demonstrated to be good at solving control problems with low costs in computer and monetary resources; but that was not all as FL allowed designers to use rules in common language to model control systems. FL is a generalization of traditional logic since it uses not only true or false but a continuous degree of being true or false. Thus, FL is used to model concepts into continuous variables. Therefore, it is possible to translate human rules for solving a problem into precise controllers that use continuous variables.

Learning Classifier Systems (LCS) [13] are systems that can learn and adapt when acting. They are compounded by a set of objects called classifiers that can be simply if-then rules. The LCS decides which rules are good and which ones are not by RL. Rules are changed using a GA. LCSs are an approach that combines two ideas: learning and adaptation. Therefore, each classifier represents an association among a set of states of the problem, one action and a parameter that represents the prediction of the reward to be obtained. The GA does not act replacing all of the classifiers but just one each time. Because of RL works in problems that are defined in discrete states with discrete actions, LCS also works with discrete problems. Many efforts have been done to make LCS works with continuous problems [34, 15, 36, 28]. One of them is the introduction of fuzzy logic [31, 20, 4].

1.1 Motivation

In the last years many techniques, like Neural Networks, Reinforcement Learning, Genetic Algorithms and Learning Classifier Systems, have risen in the Machine Learning field. These techniques learn in many different ways. For instance, they use examples with supervision or reward, or the minimization or maximization of a fitness function. But LCSs have called the attention of researchers in the area because they are capable of learning the structure of a problem without the introduction of a model. This learning is achieved by the use of *classifiers*, that are simple condition-action rules that map a set of states from the problem to one or many possible actions. This means that the solution of a problem is created by the combination of many rules over the state space of the problem. Furthermore, they do not cooperate but compete. In this way, there are many rules that propose a possible action for the observed state. These rules are modified by a GA which replaces the bad ones by possibly good rules. This GA uses as fitness function the amount of reward the system has received when acting. Classifiers can learn general rules that involve many states that are classified as similar ones. Therefore, when it is possible, these general rules decrease the amount of knowledge

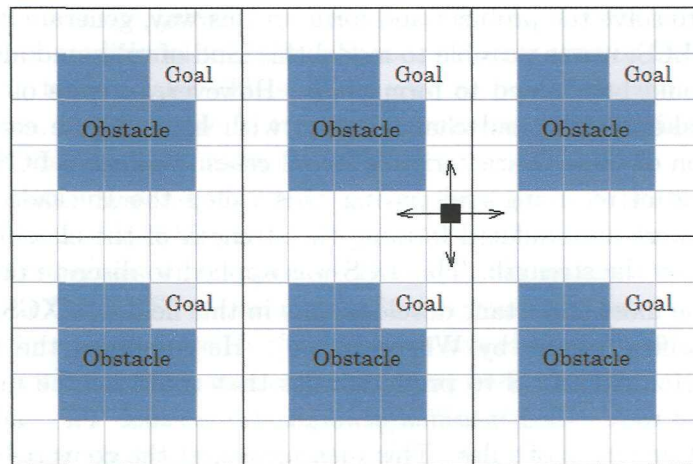
the system needs to act. LCSs are also capable of learning at the same time that they are exploring. This characteristic is very important because humans learn at the same time they experience the environment; hence, this property is very desirable in artificial intelligence systems.

The foundations of LCSs, which were inspired by the ideas of Induction Theory [14], were presented by Holland [13]. The Induction Theory tries to model how minds work. It describes that humans learn by generating general rules that cover entities with similarities. These rules are not always correct so they have to be modified. Nevertheless, these rules could be sometimes incorrect. Thus, general rules could coexist with some specific rules to solve the problem and form, in this way, general-exception rules. With the Holland's LCS, it was possible to model this kind of rules and also other mechanism where rules could be chained to form plans. However, in spite of Holland's efforts to achieve general-exception and chained rules with his LCS, he could not achieve the implementation of these characteristics for all cases. Holland's LCS learned by reward using a scheme of receiving and paying bids called the Buckade-Brigade algorithm. received Bids were accumulated forming the strength of the classifier. Classifiers were evolved based on the strength. This LCS was applied to discrete problems.

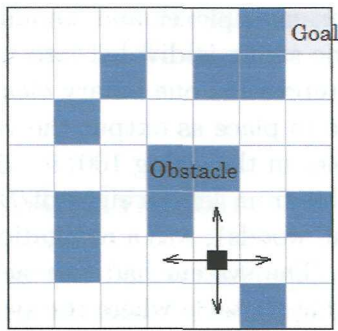
One of the most important developments in this field was XCS, an accuracy based Learning Classifier System by Wilson [33, 9]. He combined the ideas of Q-learning [24, 27] with Holland's LCS to produce rules that could be the most general ones as possible. These were called *maximal generalizations rules*. The main contribution was the concept of *accuracy* of rules. This idea measured the convergence of the expected reward of the rules. Thus, they could evolve to find general rules without any errors. XCS worked well in simple environments like the n -multiplexer and woods1. The n -multiplexer consists of binary strings as states. The string is divided into two subsets A and B . A combination of 0s and 1s in subset A represents one binary element of the subset B . Therefore, the actions of the problem are to place as output the value of the bit in B that was selected by the set A . For example: in the string 100110, $A = 10$ and $B = 0110$, if combination 01 represents the third bit from left to right of B , then the output is the action of placing 1. The other problem, woods1, was a navigation problem in a periodic two-dimensional grid space of 5×5 . The system had to reach the goal that was in the corner of the objects. There was a big obstacle where the system could not pass. The actions were go up, down, left and right. This problem is shown in Fig. 1.1.a. The obstacles are in dark gray and the goal in gray.

Another achievement came from Stolzmann [25, 7, 16] who included the Anticipatory Behavioral Control Theory to LCSs called Anticipatory Learning Classifier Systems (ACS). ACS could learn to predict its next state given its next action. Thus, ACS learns the structure of the environment by anticipations and this characteristic facilitates the creation of plans before any action in the problem. ACS was applied in simple problems like the n -multiplexer mentioned before and, on maze6 and wood14 that are also navigation tasks on two-dimensional grid spaces and are shown in Fig. 1.1.b and Fig. 1.1.c.

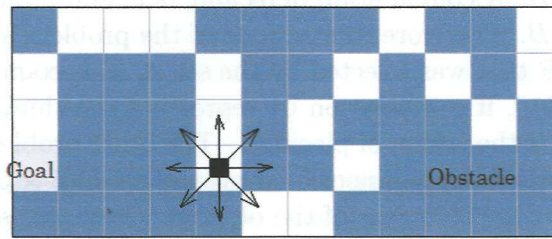
The fact that many real world problems have something to do with continuous variables instead of discrete ones have encouraged researchers to look for ways to take LCSs to continuous problems. Thus, XCS was modified to deal with continuous perceptions



a)



b)



c)

Figure 1.1: a. Woods1. b. Maze6. c. Woods14

in [26, 35, 8] to solve the real n -multiplexer but the most important modification came from Wilson [34] and was called XCSF. The XCSF only learned continuous functions. Its machinery had a change in the rule representation such that rules were activated over hyper-rectangular regions of the input space and the values of the expected payoff were calculated by adjusting a hyper-linear function by a modified delta rule. Then, the continuous function was obtained from the values learned of the expected payoff. The XCSF had only one action that was not used to generate the output of the system, therefore, it does not make rules compete. Lanzi et al. [15] applied the XCSF to continuous navigation tasks in one dimension (the continuous linear corridor problem) and two dimensions (the 2D continuous gridWorld problem). In these learning tasks, XCSF perceived continuous inputs and chose an action from a set of discrete actions. In [36] Kovacs et al. proposed three different architectures that used combinations of two XCSFs to deal with continuous inputs and outputs in a simple problem called the *frog problem*, in which the system is reset after each action. After, this problem was also tackled modifying the XCSF [28] using two GAs, one for evolving actions and the other for evolving rules. The frog problem consists of a frog placed at random over a line in the interval $[0, 1]$ and of a fly placed on that line at position 1. The frog has to jump just once to catch the fly. In this problem the actions are continuous in the interval $[0, 1]$.

Some other researchers considered that Fuzzy Logic [32] was a better option to deal with continuous variables. Therefore, two LCSs [29, 30, 31, 20] that use FL were introduced to learn functions from rewards. The first one used fix membership functions, and the second one allowed the membership functions to be changed in their shapes and positions. These FCSs used cooperation instead of competition among rules to determine outputs. This was due to FL because in FL rules are combined with others to produce outputs. Bonarini [3] introduced another framework where competition and cooperation among rules were used to learn an action function in navigation tasks with continuous reward. Continuous reward means that the system receives reward every time at every position. In other words the system is guided all the time. A more complex problem would be where the system receives reward only in the goal and otherwise nothing. Bonarini's FCS was applied [4, 18] to a CAT robot to learn how to keep itself in the center of a corridor while moving with constant speed through the corridor. The reward was a function of the distance between the robot and the center of the corridor. This FCS was tested with different learning schemes like the Bucket-Brigade Algorithm, Temporal Differences and Q-Learning. These systems produced continuous actions from a continuous perception. The problem is that it has an exponential number of internal fuzzy states with respect to the number of input variables to represent the perception of the robot. Thus, it has to form one activation set for each possible internal fuzzy state.

LCSs, in spite of being a good approach for learning due to its characteristics, do not have tools to deal with continuous problems where a set of actions to accomplish a task is needed. XCSFs [36, 28] have shown to be good at this problem without using of fuzzy logic but only in problems where just one action is required. And Bonarini's FCS [3], that has been used in real robots, has the drawback of an exponentially large internal set of fuzzy states. This is an important issue because many problems require many variables. Nevertheless, this approach has combined discrete with continuous

variables with success. Therefore, finding new mechanisms, that are capable of solving continuous problems where many actions are needed to complete a task, is still an open and relevant problem.

1.2 Related Works

The interest is solving the generalized n -Environment Problem. This problem is defined with a continuous n dimensional input space and with a set of continuous vector actions. Meaning that, the system has to perceive a continuous vector and select a continuous vector as action. A solution of the problem is to reach a certain region defined in the input space and for this the system could make more than one action. These problems are known as multi-step problems.

Kovacs et al. [36] proposed three architectures that used a group of two XCSFs to deal with a simple problem called the frog problem. The frog problem is a 1-Environment Problem with the restriction of making only one action to solve the problem. These kinds of problems are known as single step problems. The problem has a frog and a fly in a one-dimensional space in the interval $[0, 1]$. The frog has to jump just once to catch the fly and the length of that jump is in the interval $[0, 1]$. Thus, the frog defines one action vector of one dimension. Note that, this problem is one of the simplest with respect to the instances that the n -Environment covers, that in general are multi-step. Three proposed architectures were: one was based on interpolation, the second on an actor-critic paradigm, and the third on treating the action as a continuous variable homogeneous with the input. The first uses one XCSF₁ with a discretization of the actions of the problem and the other XCSF₂ is programmed to learn a continuous function based on the actions taken by the XCSF₁. Thus, this system works as an interpolation algorithm. The second uses a system XCSF₁ that makes an action based on a weight vector and the input vector, and an XCSF₂ that works as an approximation function to predict the reward to be obtained. In the third approximation introduced in the XCSF the input and the output of the problem are changed to be the input of the system. This system approaches the solution by pieces of continuous curves. QFCS works better in this problem than the approximations mentioned before; moreover, QFCS can deal with problems that more complex than the frog problem. A generalized frog problem could be the same frog in one dimension but with a fly too far to catch in just one jump. It means that to catch the fly it has to make more than one jump. This problem could be taken to the frog in two dimensions that is also more complex. QFCS can deal with these more complex problems of the frog.

Trung et al. [28] also solve this frog problem modifying XCSF by the introduction of two GAs. First, a change in the representation of classifiers was done. This change introduces the calculus of an action through a dot product of two vectors: an action vector and the input vector. This generates lines in the input-output space. The prediction values now were approximated by a plane over the input and output space and was used to extrapolate the prediction of reward out of the curve that was represented by the dot product of the action and input vectors. Therefore, one GA evolved the

action vectors of the classifiers and the other the classifiers. This approach works better than the ones proposed by Kovacs et al. [36] and it works better than QFCS but this algorithm was only tested in this simple problem.

Bonarini's approach [4, 18] introduces FL in the LCS. This FCS combines the competition and the cooperation among rules in different stages of the algorithm. When the input enters to the system, it creates a set of internal fuzzy sets based on all combinations of the input fuzzy sets of rules. Therefore, for each possible internal state a set of activation rules is created. From these activation sets, one rule per set is selected by competition. These selected rules form a fuzzy system and produce the output. This FCS was applied to a problem that combines continuous and discrete variables. One application is in the corridor problem where it is a CAT robot that has to control itself to be in the center of the corridor. The robot is moving with constant speed through the corridor. The robot received reward with respect to the distances the robot has to the center of the corridor. This problem is a navigation task with continuous reward that guides the robot. The emphasis in the problems QFCS solves is that the system only receives rewards when reaching the goal and nowhere else. For comparison with the CAT robot problem, the CAT robot problem has to receive reward only when it is in the center of the corridor. Given these conditions, it is a task much more difficult. Let us make an analogy. Imagine there is an airplane that can be anywhere in the world. Then it is required that the airplane goes to México. The airplane does not know how to reach México, and it is going to perform many tests to find México but when it reaches México, the airplane receives a reward; and each time it receives that reward it learns how to reach México. It is obvious this is very difficult to find even for a person. Giving reward from every place in the world is equivalent to guide the airplane through México, so it is easier to find a solution in this way.

1.3 Problem Statement and Context

LCSs can be classified according to the kind of problems they can solve. Such a classification is shown in Fig. 1.2. As it can be seen, the state space (input variables) of the problem can be discrete or continuous. Similarly, the action set (output variables) that defines how the transitions are among states can be discrete or continuous. Furthermore, the problem can also be one-step or multi-step. This means that to reach points defined by the problem as goals, the execution of one or more than one action is required, respectively. Thus, the solution of a multi-step problem has to be a sequence of states and actions as

$$X_0 \xrightarrow{A_0} X_1 \xrightarrow{A_1} X_2 \xrightarrow{A_2} \dots \xrightarrow{A_{n-1}} X_n \quad (1.1)$$

where X_0 is the initial state of that sequence with $n - 1$ actions and where X_n is a goal point. This sequence could be not unique. It is obvious that the most complex problem is continuous in the state space and actions, and is multi-step.

LCSs can also be classified according to the kind of approaches used. Thus, some LCSs have used FL to deal with continuous inputs while others not. Some others have introduced modifications of Q-learning as the learning algorithm used instead of the traditional Bucket-Brigade algorithm. In this way, Table 1.1 shows the classification of

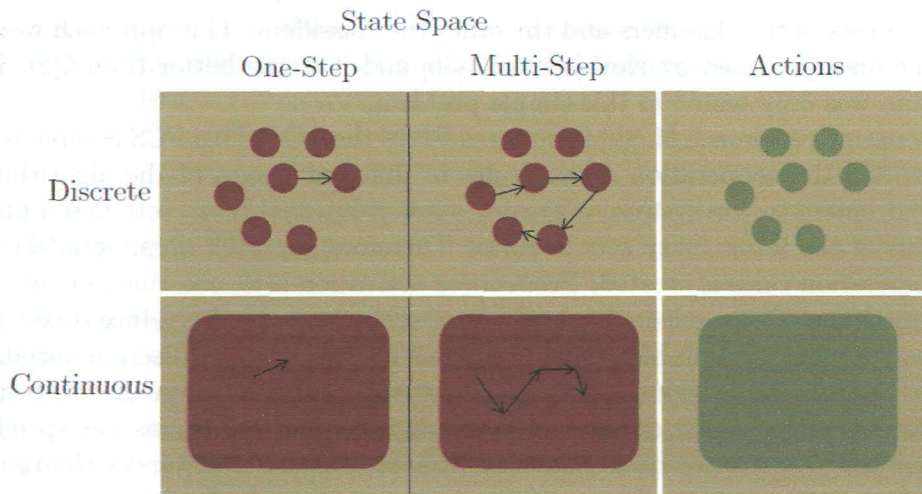


Figure 1.2: Type of problem.

the main LCSs. In theory, all of the classifiers were designed to deal with n Inputs and m Outputs. The Holland's LCS and the XCS were tested with many input and output binary variables. The XCSF was tested with a maximum of 2 input variables and with 1 discrete output variable. The others are in the table.

The problem is to find a new mechanism for an LCS that introduces Q-learning and Fuzzy Logic to solve multi-step problems with n continuous inputs and m continuous outputs. The introduction of Q-learning is because it is wished that the system can learn by reward in a multistep environment. This reward has to be given to the system only in those cases where it has acted well. Fuzzy Logic is used because it is considered the best approach to deal with continuous variables as it has been demonstrated in control systems.

1.4 Research Questions

The main questions to answer are:

- Is it possible to develop an LCS that uses Fuzzy Logic and Q-learning to solve the n -Environment Problem?
- Does this LCS have to have fixed linguistic concepts or can they be defined in the learning process?
- Is this LCS capable at least to solve the Frog Problem defined by Kovacs et al. [36]?
- Is this LCS capable of solving continuous simple navigation tasks in one and two dimensions?
- Is this LCS capable of dealing with more realistic problems like the movement of an inertial particle?

Type of LCS	Input		Output		Steps	FL	QL
	Dim.	Type	Dim.	Type			
Holland's LCS	n	D	m	D	≥ 1	×	×
Wilson's XCS	n	D	m	D	≥ 1	×	✓
Wilson's XCSF	n [2]	C	m [1]	D	≥ 1	×	✓
Wilson's 2 XCSFs	n [1]	C	m [1]	C	1	×	✓
Trung's XCSF	n [1]	C	m [1]	C	1	×	✓
Valenzuela's FCS	n [1]	C	m [1]	C	1	✓	×
Parodi's FCS	n [1]	C	m [1]	C	1	✓	×
Bonarini's FCS	n	C	m	C	≥ 1	✓	✓

Table 1.1: Classification of the LCSs. D=Discrete, C=Continuous, []=Maximum number used in experiments.

1.5 Solution Overview

QFCS [21, 22, 23] was designed to solve problems that can be defined in continuous state spaces with a set of continuous vector actions. The task in these problems has to be defined by a goal that is a region in the state space and that QFCS has to reach by means of performing more than one action. The action has the effect to change the state of the problem. The system has to learn by receiving reward only when it reaches the goal. In its abstract form, this kind of problem is called the n -Environment Problem [22]. Thus QFCS uses FL to model the continuous input and output variables of the problem, and Q-learning to learn by reinforcement. Initially, the input fuzzy sets were fixed but they introduce restrictions to what QFCS could learn; in spite of those restrictions the performance of QFCS is good. Thus, allowing modifying the input fuzzy sets introduces a new level of complexity but the results are the same.

QFCS is divided in components: classifiers, performance component, learning component and a discovery component.

- Classifiers are compound by a region in the input state space of the problem, a hyper-matrix with the same dimension of the input space and a Small Fuzzy System (SFS) defined over the activation region and that only acts there. These values of the hyper-matrix are associated with small regions in the activation region and represent a prediction of reward. These small regions are uniformly distributed. The SFSs create a continuous vector field that relates the input space with the output space and this vector field represents the actions the classifier does from each possible input vector in the activated region. The internal structure of all SFSs is the same.
- The Performance Component determines how QFCS acts taking into account the information it has acquired. Thus, when the input vector enters to QFCS, all classifiers that contain that vector inside of their activation region are activated

to form a set called $[M]$. All classifiers in $[M]$ compete to place their vector action as the output of the system. The system selects the one which has the highest prediction value associated with the small region which contains the input vector inside. This means that not all of the elements in the hyper-matrix are taken into account in this decision but also one element of the hyper-matrix for each classifier that contains in its associates small region the input vector. This is in this way because the hyper-matrices are to represent an approximation to continuous Q-function of Q-learning.

- The Learning Component acts when the system is learning. In this case QFCS decides which action to select by two mechanisms that are combined with a certain probability. The first mechanism is exploitation and the selection is similar to the selection in the performance component. The second mechanism is exploration and the selection is done taking a classifier at random. The values of the hyper-matrices in the time before are adjusted according to a similar rule found in Q-learning.
- In the Discovery Component classifiers are evolved by a GA based on the average of their hyper-matrices. Each time the GA acts only one classifier is replaced.

As it was said QFCS was designed in two different versions: one with fixed fuzzy sets and the other with unfixed fuzzy sets. The first one has the activation regions fixed. Thus, the GA does not evolve the activation regions of the classifiers but only the actions of the fuzzy rules in the SFS. The input fuzzy sets are also fixed. Classifiers belong to each possible activation region with the same probability so there are about the same number of classifiers containing the same activation region. This QFCS was tested with 5 different instances of the n -Environment Problem which have different levels of difficulty: navigation tasks in one and two dimensions and with an inertial particle problems. Navigation task in one dimension is simple but is multi-step in a continuous space with a continuous range of actions that determines the movements of the system. A difference with the frog problem is that the navigation task in one dimension has more possible actions to perform per each position to be capable of reaching the goal while the frog problem only has one per each position. Imagine the frog is in position 3 and the fly in position 4 the size of the jump is only 2. In the navigation task there are more possible combinations of giving the jumps to reach the goal. An optimal solution would be to reach the goal with the less jumps possible. Navigation task in two dimensions introduces more difficulty due to dimensionality and because of the actions that now are a vector. This vector is continuous because all directions are possible and with all magnitude less than a defined parameter. In this way the action of the system has to be directed to the goal. The system has to learn in which direction to move and with which magnitude. The inertial particle problem is more difficult because of the transition function. This transition function does not allow the system to move in each possible direction but it has a preference to one particular direction because of the inertia.

In the second QFCS, the one with the unfixed fuzzy sets, the activation regions are not fixed anymore. It determines the width and the position of the input fuzzy

sets because the SFS is defined over the activation region. The GA evolves the activation regions and the action parts of the fuzzy rules of the SFS. A new hyper-matrix is introduced that saves the predictions of the reward but in a normalized way. This normalization is done because now the classifiers have different activation regions. This makes classifiers evolve to those parts where the Q-function is higher. This does not happen in QFCS with fixed fuzzy sets because the activation regions cover all the input state space and they do not change. But classifiers move towards the goal if the activation regions are free. The normalization process is to avoid this and maintain all classifiers covering all the input state space. Allowing the changing of the activation region was to overcome the limitations QFCS has due to fixed activation regions.

In this way, the key idea of the solution model called QFCS is to solve the problem called the n -Environment Problem, which is indeed a continuous state space problem. Therefore, a solution means to move from one point to those points that are represented as goal ones. Thus, QFCS has to perform more than one action to reach some goal point. One important characteristic of this problem is that it is *Markovian*. Roughly speaking, it means that in each possible state, there should not be two different correct actions because this situation could make the system not learn the correct one. In the n -Environment Problem where the actions correspond to a navigation task, there are more than one possible optimal actions for each state. Since these many actions for the same state represent similar displacements, the problem continues to be *Markovian*.

The way QFCS solves the problem is by having classifiers represent hyper-curves in the combined state-action space. These hyper-curves are represented by the SFS that each classifier has. They are SFS because they have only 2^n fuzzy rules where n is the dimension of the state space. Thus, classifiers map a set of continuous states to a set of continuous actions. These continuous actions are actually vectors. QFCS represents the Q-function of Q-learning only in the points over these hyper-curves and uses a modification of Q-learning to learn this Q-function. The Genetic Algorithm evolves these hyper-curves maximizing the average of their Q-values or their normalizations.

1.6 Main Contributions

The main contributions of this research are:

- The introduction of a new Fuzzy LCS that is able to deal with multi-step continuous problems (The n -Environment Problem) called QFCS.
- QFCS introduces the use of fuzzy systems with a few fuzzy rules as the main representation of classifiers. Thus, with fuzzy systems as classifiers it is possible to associate a set of continuous states with a set of continuous vectors. Therefore each classifier represents a continuous vector field. This property is a novelty to traditional FCSs from literature where single fuzzy rules are used in each classifier.
- The approach also introduces a modified Q-learning as the main algorithm for learning. Q-learning makes QFCS solve problems by reinforcement.

- QFCS uses hyper-matrices to learn an approximation to the real Q-Function. This property is also a novelty to traditional FCSs from literature where one value is learned per fuzzy rule. The problem with traditional FCSs is that they use a single parameter to approximate the Q-function. But fuzzy rules work in combination with other fuzzy rules that are not always the same ones in their neighborhoods. So, that value is not representative of the Q-function in those neighborhoods. Therefore, the introduction of the hyper-matrix in each fuzzy system is to learn directly the Q-function over the input state space. That is why; QFCS associates the hyper-matrices to small region in the activation regions of the classifiers. Therefore, it is very important to remark that QFCS is trying to learn the Q-function.
- Classifiers in QFCS can compete to be the better ones as in traditional LCSs. On the other hand, contrary to what is done in XCS and XCSF, QFCS solves the problem without the accuracy concept. It uses the strength that is taken by averaging the Q-values in the hyper-matrix or averaging their normalizations.
- QFCS does not find out the mapping (Q-function) on the complete state-action space but only on those regions where the mapping is important to make decisions.
- QFCS does not care which transition function the n -Environment is using.

1.7 Dissertation Organization

This dissertation is organized in 7 chapters. These chapters describe all of the theories and concepts needed to develop the solution model proposed with the definition of the testing problems used and the results obtained. Therefore, chapter 2 contains a description of Fuzzy Logic (FL). FL is needed because it forms the main representation of rules since rules contain small fuzzy systems that make the mappings between continuous inputs and outputs. Chapter 3 has an introduction to Q-learning. Chapter 4 describes the main Learning Classifier Systems mentioned before. These LCSs are Holland's LCS, XCS, XCSF, Valenzuela's FCS, Parodi and Bonelli's FCS and Bonarini's FCS. The review of these LCSs is necessary for a better comprehension of this kind of systems. Chapter 5 explains the model proposed, called QFCS; a mathematical description of QFCS with some simple examples accompanied for three dimensional figures to make better the understanding of the system is presented. Chapter 6 introduces the main problem for which QFCS was designed, the n -Environment Problem. Five instances of this problem are described and discussed. World_a^{1D} , World_a^{2D} and World_b^{2D} are navigation tasks in one and two dimensional spaces, Particle_a^{1D} and Particle_b^{1D} are more realistic problems related to the movement of an inertial particle. Chapter 7 describes the results obtained by the experimentation with QFCS over the testing problems mentioned in chapter 6. Finally, Chapter 8 has the conclusions and future work.

Chapter 2

Fuzzy Logic

Control Theory (CT) [19] has been the other side of Artificial Intelligence (AI), is the one that tackles problems that involve continuous variables. Typically, the problems to solve are defined by continuous models based mainly on systems of Differential Equations. Control systems are very precise but are expensive and complex.

Fuzzy Logic (FL) was introduced by Zadeh [37] in 1965. This theory introduces the use of linguistic concepts defined over continuous variables. In this way, FL is capable of reasoning with concepts and of making relations among continuous variables. The importance of FL in control theory is because FL showed to be suitable to be applied in control problems. The main reason was that a control problem could be defined with simple linguistic rules. This makes designers not to deal with complex models that involve systems of complex Differential Equations Systems and allows them to use the experience of other people more knowledgeable in control tasks. Another remark is that a FL controller was cheaper and faster than those from control theory.

Because FL is able to model concepts that have something to do with continuous variables, like temperature, AI researchers have used it to tackle problems that somehow are related to this kind of variables. Many problems than humans can solve are continuous because nature is continuous like a fly flying in the three-dimension space, a human walking on the street, etc. Moreover, FL is used to reason with ambiguous concepts. This is something that humans do very well.

Learning Classifier Systems (LCS) Researchers have also used FL to make LCSs deal with problems where continuous variables are involved. Thus, the LCS proposed in this dissertation as a solution model uses FL as the main representation of the continuous variables involved. These fuzzy systems are the core of the classifiers. As fuzzy systems represent functional relationships among continuous variables, each classifier actually represents a vector field.

This chapter describes briefly FL theory to form fuzzy systems. This is very important because the processes showed in this chapter are used to model classifiers in QFCS.

2.1 Fuzzy Rules

FL [37, 32] is a generalization of Classic Logic that can deal with continuous values of truth. For example, instead of saying something is true or false, in FL it could be said something is 70% true and 30% not true. In this way, FL has been used to represent relationships among continuous variables. These relationships are created by a set of rules of the type IF-THEN that act over some linguistic variables defined as

$$\mathcal{X}_1, \dots, \mathcal{X}_n, \mathcal{A}_1, \dots, \mathcal{A}_m. \quad (2.1)$$

These linguistic variables match with their corresponding continuous variables

$$x_1, \dots, x_n, a_1, \dots, a_m \quad (2.2)$$

that are differentiated in two types, x and a . The former are related with the conditional part of rules and represent inputs and the latter with the consequent part of rules and represent outputs. Linguistic variables can take a series of linguistic values. These linguistic values are called fuzzy sets and are defined as

$$\begin{aligned} \mathcal{X}_1 &\in \{X_{11}, X_{12}, \dots, X_{1N_1}\}, \\ &\vdots \\ \mathcal{X}_n &\in \{X_{n1}, X_{n2}, \dots, X_{nN_n}\}, \\ \mathcal{A}_1 &\in \{A_{11}, A_{12}, \dots, A_{1M_1}\}, \\ &\vdots \\ \mathcal{A}_m &\in \{A_{m1}, A_{m2}, \dots, A_{mM_m}\}. \end{aligned} \quad (2.3)$$

With these assignments and logic operators, it is possible to create rules that involve linguistic variables as shown in the next examples:

$$\begin{array}{ll} \text{IF } [\mathcal{X}_1 = X_{13}] & \text{THEN } [\mathcal{A}_3 = A_{32}], \\ \text{IF } [\mathcal{X}_3 = X_{34} \wedge \mathcal{X}_5 = X_{53}] & \text{THEN } [\mathcal{A}_2 = A_{22}]. \end{array} \quad (2.4)$$

Because these rules represent relationships of continuous variables, FL assigns them continuous functions that indicate their continuous truth values. These functions are called membership functions. To do this, first it assigns membership functions to linguistic values and then it applies logic operations over those membership functions. To show how this is done, the membership functions of the linguistic values and fuzzy logic operations will be analyzed next to the point where the membership function of fuzzy rules can be defined.

2.2 Membership Functions

Each linguistic value is represented as a membership function $\mu_X(x)$ (or $\mu_A(a)$) over the corresponding continuous variable range, where X (or A) represents the linguistic value and x (or a) the corresponding continuous variable. Figure 2.1 shows an example of some possible membership functions of the fuzzy sets of a linguistic variable of definition 2.3.

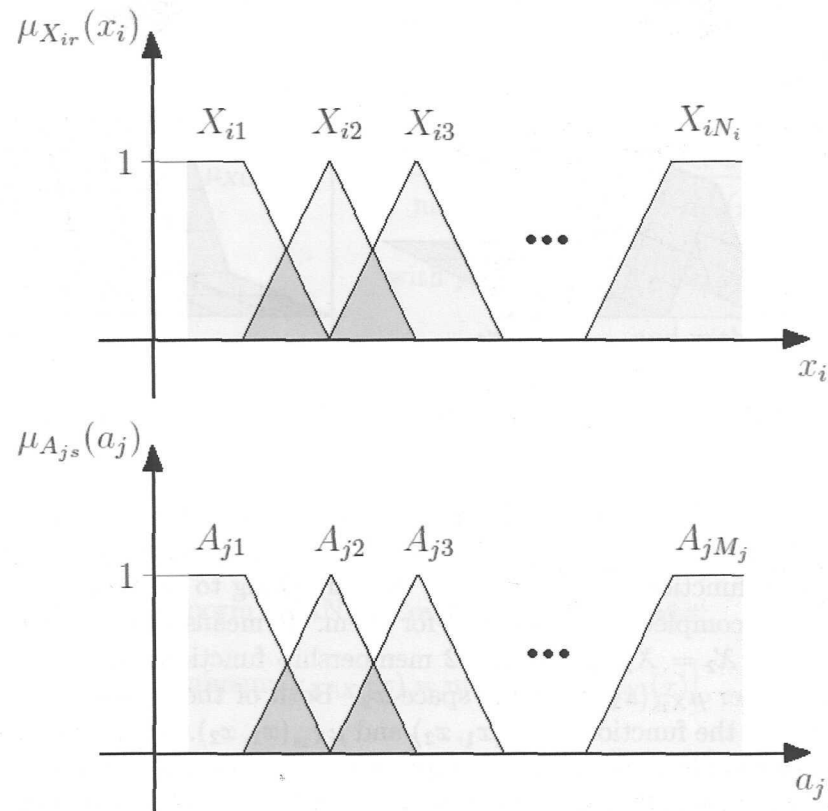


Figure 2.1: Membership functions of linguistic values

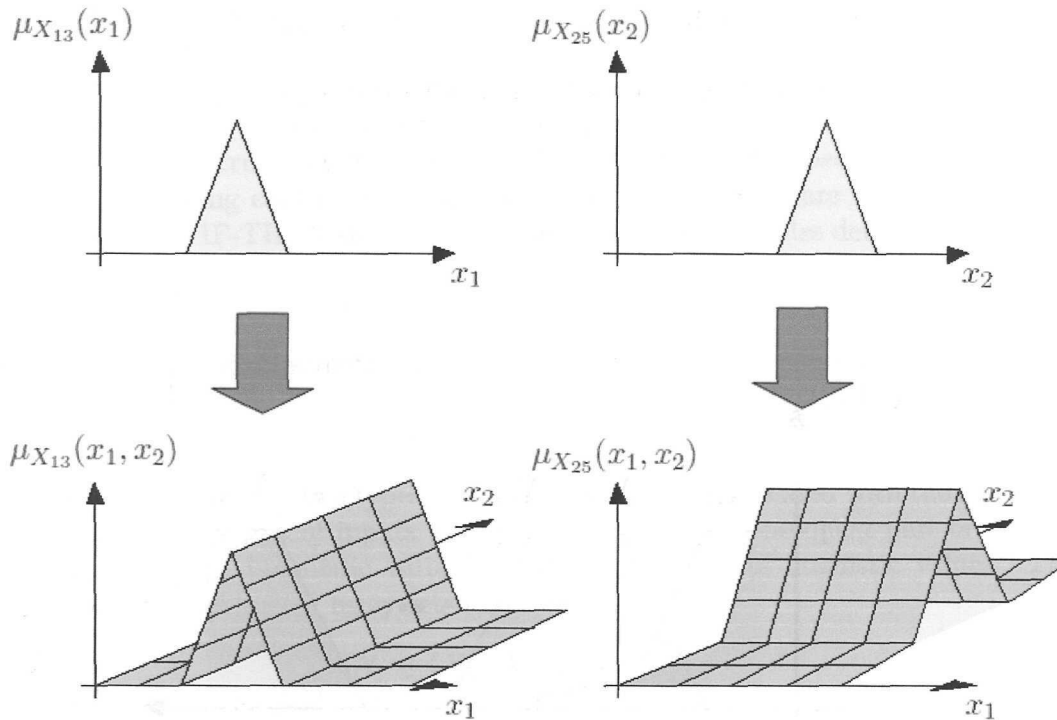


Figure 2.2: Cylindrical extensions of $\mu_{X_{13}}(x_1)$ and $\mu_{X_{25}}(x_2)$

Each membership function $\mu_X(x)$ (or $\mu_A(a)$) takes values in the interval $[0, 1]$, where 1 defines 100% true and 0 defines 0% true. Before applying a fuzzy logic operation, the membership functions of the elements that are going to be operated on have to be defined over the complete space defined for them. It means that, with the expression ($\mathcal{X}_1 = X_{13}$ AND $\mathcal{X}_2 = X_{25}$) there are 2 membership functions, one $\mu_{X_{13}}(x_1)$ over the space x_1 and other $\mu_{X_{25}}(x_2)$ over the space x_2 . Both of them have to be in the same space x_1x_2 to get the functions $\mu_{X_{13}}(x_1, x_2)$ and $\mu_{X_{25}}(x_1, x_2)$. To do this, the cylindrical extension operation is applied. This simply means to add the variable missing in the membership functions. This is shown in figure 2.2 where it can be seen how membership functions expand cylindrically to the dimension missing in order to cover the entire domain.

2.3 Fuzzy Operations

Traditional fuzzy operations are union, intersection, and complement that can be seen as an OR, AND and NOT operations, respectively. Union and intersection are also known as T-norm and S-norm, respectively. There are also implication and inference operations that are used for defining the relationships of rules and inferring from them. There are many ways of defining these operations and some of them will be mentioned.

For union, S-norm or OR (Disjunction) there exist:

$$\text{Maximum: } \mu_{X \cup X'}(x) = \max [\mu_X(x), \mu_{X'}(x)], \quad (2.5)$$

$$\text{Drastic Sum: } \mu_{X \cup X'}(x) = \begin{cases} \mu_X(x), & \text{if } \mu_{X'}(x) = 0; \\ \mu_{X'}(x), & \text{if } \mu_X(x) = 0; \\ 1, & \text{otherwise,} \end{cases} \quad (2.6)$$

$$\text{Algebraic Sum: } \mu_{X \cup X'}(x) = \mu_X(x) + \mu_{X'}(x) - \mu_X(x)\mu_{X'}(x), \quad (2.7)$$

$$\text{Dombi: } \mu_{X \cup X'}(x) = \frac{1}{1 + \left[\left(\frac{1}{\mu_X(x)} - 1 \right)^{-\lambda} + \left(\frac{1}{\mu_{X'}(x)} - 1 \right)^{-\lambda} \right]^{-1/\lambda}} \quad (2.8)$$

with $\lambda \in (0, \infty)$,

$$\text{Dubois-Prade: } \mu_{X \cup X'}(x) = \frac{\mu_X(x) + \mu_{X'}(x) - \mu_X(x)\mu_{X'}(x) - f_{XX'}}{\max [1 - \mu_X(x), 1 - \mu_{X'}(x), \alpha]} \quad (2.9)$$

$$\text{with } f_{XX'} = \min [\mu_X(x), \mu_{X'}(x), 1 - \alpha]$$

and with $\alpha \in (0, 1)$,

$$\text{Yager: } \mu_{X \cup X'}(x) = \min \left[1, \left(\mu_X(x)^w + \mu_{X'}(x)^w \right)^{1/w} \right] \quad (2.10)$$

with $w \in (0, \infty)$,

$$\text{Einstein Sum: } \mu_{X \cup X'}(x) = \frac{\mu_X(x) + \mu_{X'}(x)}{1 + \mu_X(x)\mu_{X'}(x)}. \quad (2.11)$$

For intersection, T-norm or AND (Conjunction) there exist:

$$\text{Minimum: } \mu_{X \cap X'}(x) = \min [\mu_X(x), \mu_{X'}(x)], \quad (2.12)$$

$$\text{Drastic Product: } \mu_{X \cap X'}(x) = \begin{cases} \mu_X(x), & \text{if } \mu_{X'}(x) = 1; \\ \mu_{X'}(x), & \text{if } \mu_X(x) = 1; \\ 0, & \text{otherwise,} \end{cases} \quad (2.13)$$

$$\text{Einstein Product: } \mu_{X \cap X'}(x) = \frac{\mu_X(x)\mu_{X'}(x)}{2 - \left(\mu_X(x) + \mu_{X'}(x) - \mu_X(x)\mu_{X'}(x) \right)}, \quad (2.14)$$

$$\text{Yager: } \mu_{X \cap X'}(x) = 1 - \min \left[1, \left((1 - \mu_X(x))^w + (1 - \mu_{X'}(x))^w \right)^{1/w} \right] \quad (2.15)$$

with $w \in (0, \infty)$,

$$\text{Dombi: } \mu_{X \cap X'}(x) = \frac{1}{1 + \left[\left(\frac{1}{\mu_X(x)} - 1 \right)^\lambda + \left(\frac{1}{\mu_{X'}(x)} - 1 \right)^\lambda \right]^{1/\lambda}} \quad (2.16)$$

with $\lambda \in (0, \infty)$,

$$\text{Dubois-Prade: } \mu_{X \cap X'}(x) = \frac{\mu_X(x)\mu_{X'}(x)}{\max[\mu_X(x), \mu_{X'}(x), \alpha]} \quad (2.17)$$

with $\alpha \in (0, 1)$,

$$\text{Algebraic Product: } \mu_{X \cap X'}(x) = \mu_X(x)\mu_{X'}(x). \quad (2.18)$$

For complement or NOT (Negation) there exist:

$$\text{Basic: } \mu_{\bar{X}}(x) = 1 - \mu_X(x), \quad (2.19)$$

$$\text{Sugeno: } \mu_{\bar{X}}(x) = \frac{1 - \mu_X(x)}{1 + \lambda\mu_X(x)} \text{ with } \lambda \in (-1, \infty), \quad (2.20)$$

$$\text{Yager: } \mu_{\bar{X}}(x) = \left(1 - \mu_X^w(x)\right)^{1/w} \text{ with } w \in (0, \infty). \quad (2.21)$$

In the operations of union, intersection, and complement defined above it is supposed that X and X' are linguistic values of the linguistic variable \mathcal{X} that corresponds to the continuous variable x . It is not necessary to make the cylindrical extensions for the membership functions involved because they are in the same space, but if these operations are applied to two fuzzy sets that belong to different variables then the cylindrical extensions have to be done first, and then the corresponding operation.

For implications there exist:

$$\text{Dienes-Rescher: } \mu_{X \rightarrow A}(x, a) = \max[1 - \mu_X(x), \mu_A(a)], \quad (2.22)$$

$$\text{Lukasiewicz: } \mu_{X \rightarrow A}(x, a) = \min[1, 1 - \mu_X(x) + \mu_A(a)], \quad (2.23)$$

$$\text{Zadeh: } \mu_{X \rightarrow A}(x, a) = \max[\min(\mu_X(x), \mu_A(a)), 1 - \mu_X(x)], \quad (2.24)$$

$$\text{Gödel: } \mu_{X \rightarrow A}(x, a) = \begin{cases} 1, & \text{if } \mu_X(x) \leq \mu_A(a); \\ \mu_A(a), & \text{otherwise,} \end{cases} \quad (2.25)$$

$$\text{Mamdani's Minimum: } \mu_{X \rightarrow A}(x, a) = \min[\mu_X(x), \mu_A(a)], \quad (2.26)$$

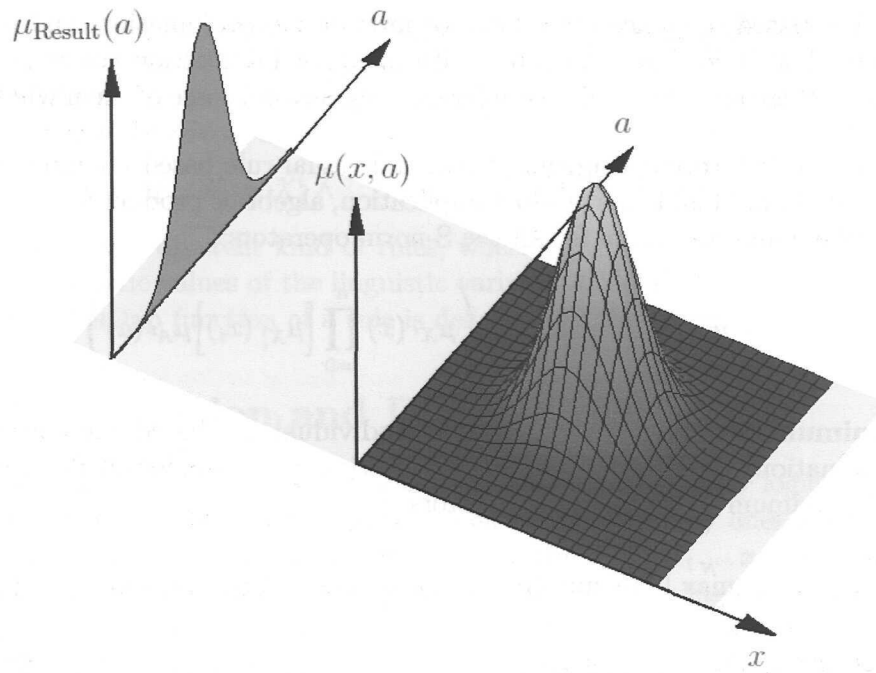
$$\text{Mamdani's Product: } \mu_{X \rightarrow A}(x, a) = \mu_X(x)\mu_A(a). \quad (2.27)$$

Since an implication represents a relationship among the variables involved, it is supposed that there are two continuous variables x and a that are represented by their corresponding linguistic variables \mathcal{X} and \mathcal{A} . X and A are fuzzy sets that belong to \mathcal{X} and \mathcal{A} , respectively, so the operations represent the membership function $\mu_{X \rightarrow A}(x, a)$ of the implication IF ($\mathcal{X} = X$) THEN ($\mathcal{A} = A$). In order to calculate the membership function of the implications, the cylindrical extension of the membership function of each fuzzy set on the space xa has to be done first.

For inference there exist:

$$\text{Generalized Modus Ponens: } \mu_{A'}(a) = \sup_{x \in U} \Gamma[\mu_{X'}(x), \mu_{X \rightarrow A}(x, a)], \quad (2.28)$$

$$\text{Generalized Modus Tollens: } \mu_{X'}(x) = \sup_{a \in V} \Gamma[\mu_{A'}(a), \mu_{X \rightarrow A}(x, a)], \quad (2.29)$$

Figure 2.3: Projection of $\mu(x, a)$ on a .

Generalized Hypothetical

$$\text{Syllogism: } \mu_{X \rightarrow Z'}(x, z) = \sup_{a \in V} T[\mu_{X \rightarrow A}(x, a), \mu_{A' \rightarrow Z}(a, z)], \quad (2.30)$$

where U and V represents the domain of discourse of x and a respectively, T represents the T-norm that is the intersection operation, and X' and A' in the first two inferences are the fuzzy sets that represent fuzzy facts needed for the inferences. The operation $\sup_{x \in U} \mu(x, a)$ represents the projection of the membership function $\mu(x, a)$ over a obtaining a new membership function defined only on space a . This operation is defined as

$$\mu_{\text{Result}}(a) = \sup_{x \in U} \mu(x, a) = \max_{x \in U} \mu(x, a), \quad (2.31)$$

where $\mu_{\text{Result}}(a)$ is the membership function only defined in the space a . Figure 2.3 shows how this operation works. As it can be seen, the greatest values on x for each a value is selected.

In FL, rules have to be combined in such a way that they cooperate to infer something; this is called the *inference engine*. The type of cooperation is defined by the way the implications are combined and the way the inference operation is applied. This can be done either, by combining all membership functions of implication through a union or intersection operation to obtain only one implication rule that represents all of them and then with this one apply the inference operation with some fuzzy fact to obtain the inferred fuzzy set, or by applying the inference operation to each implication with some fuzzy fact first, and then combining the resulting membership functions through a union or intersection operation to obtain the inferred fuzzy set. The former is called

Composition Based Inference with union or intersection combination, and the latter is called Individual-Rule Based Inference with union or intersection combination. With this in mind, there are many different inference engines and some of them will be defined.

- **Product Inference Engine.** It uses individual-rule based inference with union combination, Mamdani's product implication, algebraic product for all the T-norm operators and maximum for all the S-norm operators:

$$\mu_{A'}(a) = \max_{l=1}^N \left[\sup_{\vec{x} \in U} \left(\mu_{X'}(\vec{x}) \prod_{i=0}^n [\mu_{X_i^l}(x_i)] \mu_{A^l}(a) \right) \right]. \quad (2.32)$$

- **Minimum Inference Engine.** It uses individual-rule based inference with union combination, Mamdani's minimum implication, minimum for all T-norm operators and maximum for all S-norm operators:

$$\mu_{A'}(a) = \max_{l=1}^N \left[\sup_{\vec{x} \in U} \min \left[\mu_{X'}(\vec{x}), \mu_{X_1^l}(x_1), \dots, \mu_{X_n^l}(x_n), \mu_{A^l}(a) \right] \right]. \quad (2.33)$$

- **Lukasiewicz Inference Engine.** It uses individual-rule based inference with intersection combination, Lukasiewicz's implication and minimum for all T-norm operators:

$$\mu_{A'}(a) = \min_{l=1}^N \left[\sup_{\vec{x} \in U} \min \left(\mu_{X'}(\vec{x}), 1 - f_A + \mu_{A^l}(a) \right) \right], \quad (2.34)$$

$$\text{with } f_A = \min \left[\mu_{X_1^l}(x_1), \dots, \mu_{X_n^l}(x_n) \right].$$

- **Zadeh Inference Engine.** It uses individual-rule based inference with intersection combination, Zadeh implication and minimum for all T-norm operators:

$$\mu_{A'}(a) = \min_{l=1}^N \left[\sup_{\vec{x} \in U} \min \left(\mu_{X'}(\vec{x}), \max \left[f_A, 1 - f_B \right] \right) \right], \quad (2.35)$$

$$\text{with } f_A = \min \left[\mu_{X_1^l}(x_1), \dots, \mu_{X_n^l}(x_n), \mu_{A^l}(a) \right],$$

$$\text{and with } f_B = \min \left[\mu_{X_1^l}(x_1), \dots, \mu_{X_n^l}(x_n) \right].$$

- **Dienes-Rescher Inference Engine.** It uses individual-rule based inference with intersection combination, Dienes-Rescher's implication and minimum for all T-norm operators:

$$\mu_{A'}(a) = \min_{l=1}^N \left[\sup_{\vec{x} \in U} \min \left(\mu_{X'}(\vec{x}), \max \left[1 - f_A, \mu_{A^l}(a) \right] \right) \right], \quad (2.36)$$

$$\text{with } f_A = \min \left(\mu_{X_1^l}(x_1), \dots, \mu_{X_n^l}(x_n) \right).$$

In the above definitions, $\mu_{X'}(\vec{x})$ is the membership function of a fuzzy fact X' defined in the input space $U = x_1 \dots x_n$ where \vec{x} is a vector in it and $\mu_{A'}(a)$ is the membership function of the fuzzy set A' inferred by the inference engine. l denotes the number of the rule and there are N of them. Rules used here have the form

$$\text{IF } [\mathcal{X}_1 = X_1^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A} = A^l] \quad (2.37)$$

that can represents different kind of rules, where $X_i^l \in \{X_{i1}, X_{i2}, \dots, X_{iN_i}\}$ represents one of the linguistic values of the linguistic variable \mathcal{X}_i defined for the l -th rule. In this way the membership function of a rule is defined over the space $x_1 \dots x_n a$.

2.4 Fuzzification and Defuzzification

Now, there is a formalism to reason with the foundations of fuzzy logic, but two more mechanisms that connect the continuous values with the fuzzy ones are needed. One is to transform the input vector \vec{x}_{Input} defined in the space $x_1 \dots x_n$ of the variables x_i into a fuzzy fact X' , and the other one is to get the continuous value a_{Output} from the inferred fuzzy set A' after the inference is done. The first is known as *fuzzification* and the second as *defuzzification*. There are many ways of fuzzification and defuzzification and some of them will be mentioned.

For fuzzification there exist:

$$\text{Singleton: } \mu_{X'}(\vec{x}) = \begin{cases} 1, & \text{if } \vec{x} = \vec{x}_{\text{Input}}; \\ 0, & \text{otherwise,} \end{cases} \quad (2.38)$$

$$\text{Triangular: } \mu_{X'}(\vec{x}) = \begin{cases} \text{T}[f_1(x_1), \dots, f_n(x_n)], & \text{if } |x_i - x_{\text{Input}_i}| \leq b_i; \\ 0, & \text{otherwise;} \end{cases} \quad (2.39)$$

$$\text{with } f_i(x_i) = \left[1 - \frac{|x_i - x_{\text{Input}_i}|}{b_i} \right],$$

$$\text{Gaussian: } \mu_{X'}(\vec{x}) = \text{T} \left[e^{-\left(\frac{x_1 - x_{\text{Input}_1}}{\alpha_1}\right)^2}, \dots, e^{-\left(\frac{x_n - x_{\text{Input}_n}}{\alpha_n}\right)^2} \right]. \quad (2.40)$$

For defuzzification there exist:

$$\text{Center of Gravity: } a_{\text{Output}} = \frac{\int a \mu_{A'}(a) da}{\int \mu_{A'}(a) da}, \quad (2.41)$$

$$\text{Center Average: } a_{\text{Output}} = \frac{\sum_{l=1}^N \bar{a}^l w_l}{\sum_{l'=1}^N w_{l'}}, \quad (2.42)$$

$$\text{Maximum: } a_{\text{Output}} = \sup_{a \in V} \mu_{A'}(a), \quad (2.43)$$

where \bar{a}^l and w_l represent the center and the height of the fuzzy set A' .

2.5 General Discussion

Each classifier in QFCS has a Small Fuzzy System (SFS). This SFS is defined over the activation region of the classifier. The SFS defines a relationship between the input and the output variables. In general, QFCS has n input variables and m output variables, so, the SFS represents a continuous vector field. The Fuzzy Systems shown in this chapter make a relationship between many input variables and one output variable representing a scalar function, not a vector field. Therefore, to obtain a vector field, it is necessary to joint m equal fuzzy systems one per each output variable differing only in the fuzzy actions.

The components of the vector field are each one a scalar function over the input variables. This scalar functions are functions of n variables. Thus, if $n > 3$, the functions are known as hyper-functions. This is very important because those hyper-functions define some points in the joint input-output space that are related by the fuzzy system. It is over those points where QFCS learns the Q-function through a hyper-matrix in n dimensions associated to those points defined by the hyper-functions.

Therefore, each classifier produces a vector by its SFS. This SFS has as input only two fuzzy sets with triangular membership functions and as output singletons that are going to be defined with precision in Chapter 5.

2.6 Summary

Fuzzy Logic associates linguistic variables to continuous variables. These linguistic variables have linguistic concepts. The linguistic concepts are associated with membership functions forming a fuzzy set. Using fuzzy operation is possible to form fuzzy rules that make relationships among linguistic variables.

The fuzzy operations are: AND, OR, NOT, Implications, Inference and Inference Machines. These operations take fuzzy values and operate over their membership functions. Implication represents a fuzzy rule. The inference process is carried out combining an implication and a fuzzy value as input to entail another fuzzy value as output. A set of fuzzy rules are combined to form a fuzzy system. Inference Machines are an operation carried out by the fuzzy system that takes a fuzzy value as input and a fuzzy system to produce a fuzzy value as output. Fuzzification is an operation that creates a fuzzy value from a real value. Defuzzification is an operation that takes a fuzzy value and creates a real value.

Chapter 3

Q-Learning

It has been said that AI is reduced to the problem of learning. Everything humans do has to be learned some time before. People learn by supervision when is guided by an instructor. They learn without pervision when they discover patterns from the scratch. But maybe the most attractive way of learning is when humans modify their behavior because they expect to avoid pain or to get pleasure. This form of learning is also found in animals. The most common example is that of lab rats, where a rat is in a box with two buttoms, one for getting food and other for getting a shock. This way of learning is called Reinforcement Learning. Tasks that can be solved by reinforcement learning are defined by reaching a goal. Thus, the problem has to have a state space where some of those states are defined as goals. In this way, it is needed a transition function that is determined by the task and that says how the state changes in front of an action. In general, this transition function is stochastic. This transition function is known as a model of the environment which is characterized by the task. Over each state a value is defined that gives a prediction of the reward that is going to be obtained. These values are known as predition values.

There are three elementary solution methods in reinforcement learning: Dynamic Programming, Monte Carlo Methods and Temporal-Difference Learning. Dynamic Programming solves a system of equations called Bellman Equations numerically. These equations define the prediction values. Monte Carlo Methods solve the problem through finished experiences in the problem. The system adjusts the prediction values from those experiences. Finally, in Temporal-Difference Learning, the system learns taking information from each transition.

Q-learning belongs to Temporal-Difference Learning. In particular, Q-learning has the capability of learning the task without using a model of the environment. Instead, it introduces the use of the states and the actions of the environment without distinction as the state of the system. QFCS uses Q-learning as the heart of the learning. Thus, a description of this algorithm is required. Next sections will describe Q-learning with detail.

		Actions						
		α_1	α_2	α_3	α_4	α_5	\dots	α_m
States	χ_1							
	χ_2							
	χ_3							
	χ_4							
	χ_5							
	\vdots							
	χ_n							

Q-values

Figure 3.1: Table that represents the Q-values of $Q(x, a)$.

3.1 The Q-function

Q-learning [24, 27] is a technique that allows learning of a discrete action function through a reward mechanism. That mechanism determines what an agent has to do at every time. To do this, it learns a discrete Q-function $Q(x, a)$ over the space xa where

$$\begin{aligned} x &\in \{\chi_1, \chi_2, \dots, \chi_n\}, \\ a &\in \{\alpha_1, \alpha_2, \dots, \alpha_m\}. \end{aligned} \quad (3.1)$$

x and a mean all the states of the environment that the agent can perceive, and all the possible actions that the agent can perform. So this function represents a table of Q-values as depicted in Fig. 3.1. An agent decides which action to perform using this table. It takes the action $a(x)$ that has the maximum Q-value in the perceived state x as

$$a(x) = \arg \max_a [Q(x, \alpha_1), \dots, Q(x, \alpha_m)]. \quad (3.2)$$

This way of selecting an action is called *exploitation* because it is using the information that is in the function $Q(x, a)$.

The environment can be represented from the perspective of the perceived states and actions of the agent. In this way, the transitions of the perceived environment are represented by a graph that contains n nodes that match with the perceived states and m edges that determine the actions that the agent can do, and that connect the perceived states of the environment as shown in Fig. 3.2. The result of an action can be stochastic, so any next state could be possible with certain probability. The transition function $T(x, a, x')$ represents these probabilities where $x' \in \{\chi_1, \chi_2, \dots, \chi_n\}$. $T(x, a, x')$ defines the probability of going to state x' from state x with the action a .

The Q-function defines a new internal space where the states are formed by pairs of values (x, a) . So an agent that uses Q-learning uses an implicit internal graph that

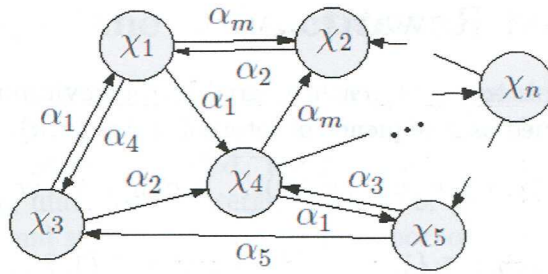


Figure 3.2: Graph that represents the transition function of the perceived environment.

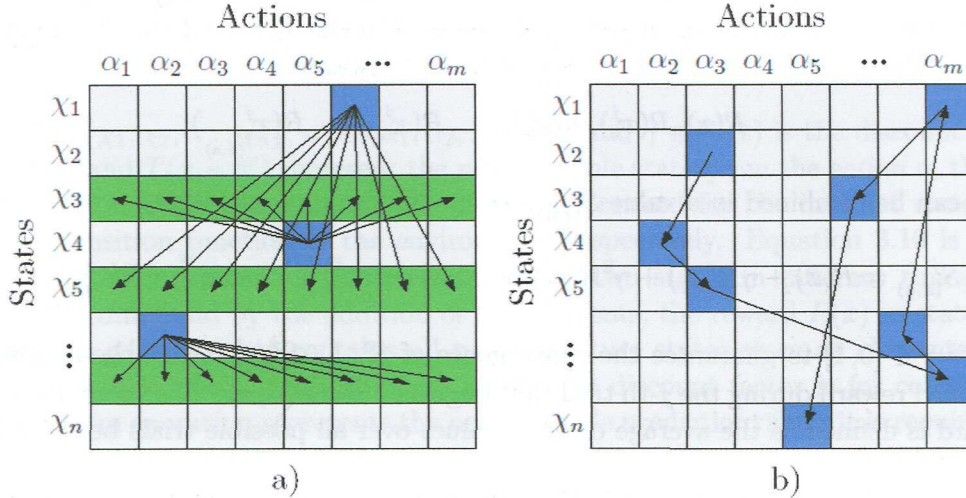


Figure 3.3: a. Three states (x, a) with their transitions. b. One sequence of actions of an agent.

defines transitions among these kinds of states as shown in Fig. 3.3a. The edges do not have labels because edges only define the next possible transition to a new state that is determined when the agent has taken its next action. A possible sequence of actions that an agent can make is depicted in Fig. 3.3b.

3.2 Reward Function

When the agent is acting in the environment in each possible perceived state it receives a reward that guides its learning. This reward is a real function defined over the perceived states $R(x)$. The goal that the agent has to reach is defined by this reward function. So the reward function can be thought as something that either rewards the agent when it has done well or punishes it when it has done badly. In this sense, the positive numbers can be seen as a reward while negative numbers as a punishment. The larger absolute number received, the larger either reward or punishment received.

3.3 Expected Reward

Each time the agent is trying to reach its goal in the environment is called a *trial*. A trial is therefore defined as a sequence of internal states (x, a)

$$T_{(x,a)}^j = (x, a), (x_1^j, a_1^j), (x_2^j, a_2^j), \dots, (x_i^j, a_i^j), \dots, (x_{I_{(x,a)}^j}^j, a_{I_{(x,a)}^j}^j); \quad (3.3)$$

where $T_{(x,a)}^j$, x_i^j , a_i^j with $i \in \{1, 2, \dots, I_{(x,a)}^j\}$ and $j \in \{1, 2, \dots, N_{(x,a)}\}$ mean the j -th trial that begins in (x, a) , the state x at the i -th internal state of the j -th trial, and the action done a at the i -th internal state of the j -th trial. There are $N_{(x,a)}$ different trials that begins in (x, a) and each one is repeated with probability $P_{(x,a)}^j$. $I_{(x,a)}^j$ is defined as the time value where the agent reach the goal in the j -th trial that begins in (x, a) . During the trial the agent is given a sequence of rewards

$$R(x), R(x_1^j), R(x_2^j), \dots, R(x_i^j), \dots, R(x_{I_{(x,a)}^j}^j); \quad (3.4)$$

that can be combined in a value $S_{(x,a)}^j$, as

$$S_{(x,a)}^j = R(x) + \gamma R(x_1^j) + \gamma^2 R(x_2^j) + \dots + \gamma^i R(x_i^j) + \dots + \gamma^{I_{(x,a)}^j} R(x_{I_{(x,a)}^j}^j) \quad (3.5)$$

where $\gamma \in [0, 1]$ to guarantee the convergence of $S_{(x,a)}^j$. $S_{(x,a)}^j$ means the summation of obtained reward during the j -th trial that begins on state (x, a) . Therefore, the expected reward is defined as the average of $S_{(x,a)}^j$ values over all possible trials beginning in state (x, a) as

$$Q(x, a) = \sum_{j=1}^{N_{(x,a)}} P_{(x,a)}^j S_{(x,a)}^j. \quad (3.6)$$

Figure 3.3b defines an example of a particular trial of the agent that begins in (χ_2, α_3) and finishes in (χ_n, α_5) . Let this trial be $T_{(\chi_2, \alpha_3)}^4$ for $j = 4$. The reward, the agent is given during that trial is

$$R(\chi_2), R(\chi_4), R(\chi_5), \dots, R(\chi_n). \quad (3.7)$$

Therefore, the value $S_{(\chi_2, \alpha_3)}^4$ is

$$S_{(\chi_2, \alpha_3)}^4 = R(\chi_2) + \gamma R(\chi_4) + \gamma^2 R(\chi_5) + \dots + \gamma^{I_{(\chi_2, \alpha_3)}^4} R(\chi_n). \quad (3.8)$$

This value $S_{(\chi_2, \alpha_3)}^4$ is one of the all possible and has a probability $P_{(\chi_2, \alpha_3)}^4$. Therefore the average over all these possible values $S_{(\chi_2, \alpha_3)}^j$ represent the Q-value of the Q-function

$$Q(\chi_2, \alpha_3) = \sum_{j=1}^{N_{(\chi_2, \alpha_3)}} P_{(\chi_2, \alpha_3)}^j S_{(\chi_2, \alpha_3)}^j. \quad (3.9)$$

Thus the Q-values represent the expected accumulated reward from the perceived state when doing the corresponding action. In this manner, it can be seen that the agent decides how to act to receive the highest expected reward.

3.4 Learning of the Q-function

To learn the Q-function, the agent has to calculate the expected reward in each state (x, a) as in Eq. 3.6. This is a little bit complicated because knowledge of all possible trials and their probabilities are needed. Furthermore, the trials and the probabilities change during learning until they converge. Therefore, the idea is to use the relationship between the internal state (x, a) and its neighborhood internal states. Thus, the expected reward $Q(x, a)$ is the immediate reward for the internal state (x, a) plus the expected discounted reward of the next internal state, assuming that the agent chooses its best action by exploitation as in Eq. 3.2. That is

$$Q(x, a) = R(x) + \gamma \sum_{x'} \left[T(x, a, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \quad (3.10)$$

where $x' \in \{\chi_1, \chi_2, \dots, \chi_n\}$, $a' \in \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ and $\gamma \in (0, 1)$ is the discount factor. x' , a' , $R(x)$ and $T(x, a, x')$ represent the next possible state given the action a , the next possible action given the state x' , the immediate reward for the internal state (x, a) and the transition function of the environment respectively. Equation 3.10 is known as the Bellman equation. This equation relates two states in different times. The equation is compound by the addition of two elements, the reward $R(x)$ in state (x, a) and another addition that contains all possible future states since all elements of the transition function that involved x' are used. The discount factor is for convergence. The maximum operation represents the next possible prediction; since it is required, the system takes those states with the maximum Q-values. Thus, Eq. 3.10 is true for all possible internal states (x, a) . Therefore, it represents a system of equations

$$\begin{aligned} Q(\chi_1, \alpha_1) &= R(\chi_1) + \gamma \sum_{x'} \left[T(\chi_1, \alpha_1, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\ Q(\chi_2, \alpha_1) &= R(\chi_2) + \gamma \sum_{x'} \left[T(\chi_2, \alpha_1, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\ &\vdots \\ Q(\chi_n, \alpha_1) &= R(\chi_n) + \gamma \sum_{x'} \left[T(\chi_n, \alpha_1, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\ Q(\chi_1, \alpha_2) &= R(\chi_1) + \gamma \sum_{x'} \left[T(\chi_1, \alpha_2, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\ Q(\chi_2, \alpha_2) &= R(\chi_2) + \gamma \sum_{x'} \left[T(\chi_2, \alpha_2, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\ &\vdots \\ Q(\chi_n, \alpha_2) &= R(\chi_n) + \gamma \sum_{x'} \left[T(\chi_n, \alpha_2, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \end{aligned}$$

$$\begin{aligned}
& \vdots \\
Q(\chi_1, \alpha_m) &= R(\chi_1) + \gamma \sum_{x'} \left[T(\chi_1, \alpha_m, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\
Q(\chi_2, \alpha_m) &= R(\chi_2) + \gamma \sum_{x'} \left[T(\chi_2, \alpha_m, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right], \\
& \vdots \\
Q(\chi_n, \alpha_m) &= R(\chi_n) + \gamma \sum_{x'} \left[T(\chi_n, \alpha_m, x') \max_{a'} \left(Q(x', \alpha_1), \dots, Q(x', \alpha_m) \right) \right]; \quad (3.11)
\end{aligned}$$

that has to be solved simultaneously for the Q-values. These equations can be also solved numerically. This technique is called *value iteration* and it is done off-line.

There is another approach called Temporal Differences (TD) [24, 27] that can be used on-line. This approach considers that the agent can use the observed transitions when acting on the environment to modify the Q-values using the next way of adjusting:

$$Q(x, a) \leftarrow Q(x, a) + \beta \left[\left(R(x) + \gamma \max_{a'} (Q(x', \alpha_1), \dots, Q(x', \alpha_m)) \right) - Q(x, a) \right], \quad (3.12)$$

where $\beta \in (0, 1)$. As it can be seen, Eq. 3.12 changes the Q-value of the state (x, a) in a fraction β of the difference from the new calculated expected reward and the expected reward on the state (x, a) . It is also shown that the transition function of the environment is not needed anymore because in the limit of a infinity number of trials, the states are visited with the probability of the transition function. Therefore, the effect is the same.

3.5 Exploitation and Exploration

When the agent is learning using TD, there are two mechanisms that are used to act. The first is the one that it was already defined before where the agent selects an action based on what it has learned using Eq. 3.2, and the second is when the agent selects an action randomly. These two different mechanisms of selection of actions are called *exploitation* and *exploration*, respectively. These mechanisms are used combined with probabilities P_E (by exploitation) and $P_R = (1 - P_E)$ (by exploration) of taking actions a_E or a_R respectively. This combination is used during learning because some better transitions can be missed by exploitation alone. These transitions can be only found with exploration.

Figure 3.4 shows an abstract view of Q-learning when learning in some points over time. There are two axes, one for the states and the other for actions (it is considered, that the states and actions are continuous for clarity). Thus, x_t is the state where the system is at time t (the states are in time order for clarity) In that state, the system selects two actions, one by exploitation $a_E(t)$ and the other by exploration $a_R(t)$. These actions in the state x_t are points with coordinates $(x_t, a_E(t))$ and $(x_t, a_R(t))$ (these

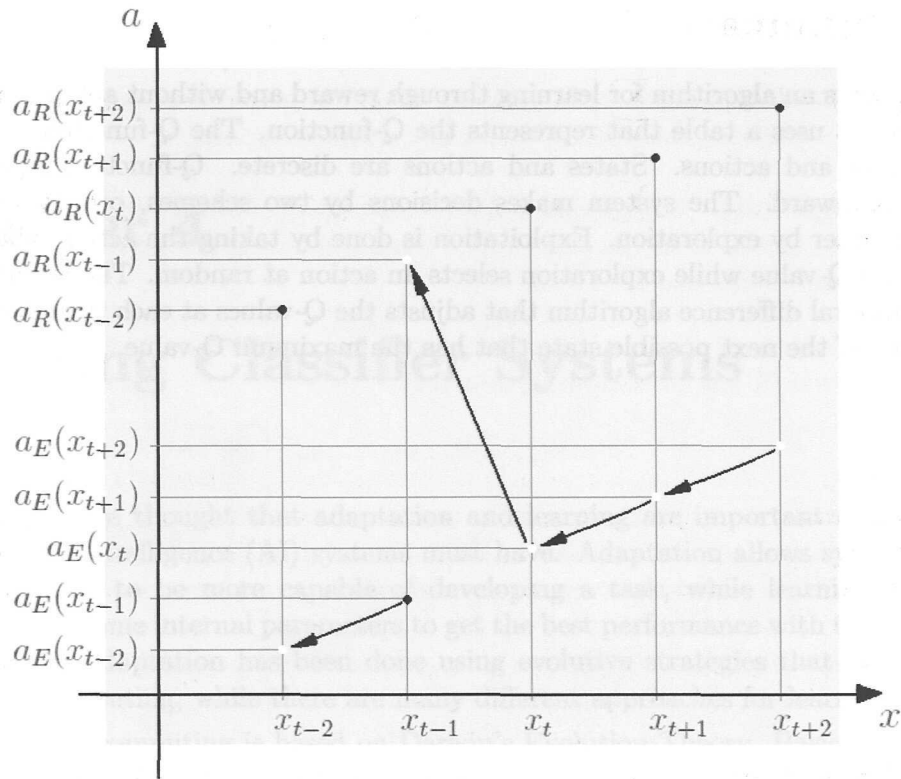


Figure 3.4: Updating of Q-values on time.

actions are in time order and separated for clarity, too). The system has to select one action from those $a_E(t)$ and $a_R(t)$ with probabilities P_E and P_R to act in the environment. This action is drawn as a point in white color. Arrows show the two states that are to be modified by Eq. 3.12. This means that the Q-values of the points obtained by exploitation are the ones used to update the Q-values of the points selected at time before (the white points).

3.6 General Discussion

QFCS uses Q-learning algorithm for learning. However, Q-learning works in problems defined as discrete while QFCS works in continuous ones. In QFCS, this problem is overcome because classifiers combine the fuzzy systems with Q-learning through the hyper-matrices. Therefore, each classifier represents hyper-functions that defined hyper-surfaces. The hyper-matrices represent the Q-values over those hyper-surfaces. In this way, each classifier tries to learn the Q-function over those hyper-surfaces. Those hyper-surfaces do not represent all of the input-output space but just a small sub-region. Since, there are many classifiers; it is possible to cover all input-output space. It is important to note that QFCS evolves classifiers moving the hyper-surfaces to those places where the Q-function is higher and because of that is that QFCS finishes with a mapping over the regions in the input-output spaces that are important for making decisions.

3.7 Summary

Q-learning is an algorithm for learning through reward and without any prior transition function. It uses a table that represents the Q-function. The Q-function is a function over states and actions. States and actions are discrete. Q-function represents the expected reward. The system makes decisions by two schemes, one by exploitation and the other by exploration. Exploitation is done by taking the action which has the maximum Q-value while exploration selects an action at random. The learning is done by a temporal difference algorithm that adjusts the Q-values at each time depending on the value of the next possible state that has the maximum Q-value.

Chapter 4

Learning Classifier Systems

Researchers have thought that adaptation and learning are important characteristics that Artificial Intelligence (AI) systems must have. Adaptation allows systems change in its structure to be more capable of developing a task, while learning allows the system adjust some internal parameters to get the best performance with that structure. Traditionally, adaptation has been done using evolutive strategies that are known as Evolutive Computing, while there are many different approaches for learning.

Evolutive computing is based on Darwin's Evolution Theory. Basically, the main idea is that there exists a set of individuals called population. This individuals form part of some generation. Generations are used to produce new generations by selecting, crossing and mutating the strongest individuals. There are many ways for carrying this out. Genetic Algorithms (GA) [12, 11, 10] are one of the approaches that uses binary strings to represent individuals.

Learning can be supervised, unsupervised and by reinforcement [2, 1, 27]. These three approaches are very different. Supervised learning trains the system with a set of known data that contain the inputs and the responses. Examples of supervised learning are pattern recognition and neural networks. With unsupervised learning the data training does not contain the responses so the system is able to discover patterns. Examples of this kind of learning are Hopfield's neural networks and decision trees. Finally, with reinforcement learning the system learns from experiences in the problem receiving reward when it has acted well and punishment when it has not.

Learning Classifier Systems (LCS) are systems that can adapt and learn from experiences. To do this, LCSs combine GAs and Reinforcement Learning. Basically, LCS has a population of rules of the type IF-THEN acting in parallel. Each rule defines an action for a subset of all the possible perceived states of the environment. This means that rules have a representation that allows generalization. These rules compete to determine the actions to be performed over the environment. Using a reward mechanism, a LCS can learn which rules are the best ones for performing a task. Rules are adapted by evolution depending on how good they are. Therefore, these characteristics make LCSs good adaptive systems that can learn from the environment without previous knowledge using reward as a guide. The following lines review the most important existing LCSs.

4.1 Hollands LCS (The first LCS)

This LCS [13] is composed by a message list, a population of rules, a performance component, a learning component, and a discovery component.

- **Message List.** It contains a set of messages that can be produced from the environment or from the other rules. Each message is a sequence of 0s and 1s. So a message looks like 01011010.
- **Population of Rules.** It has a fix set of rules $[P]$. This set is called *population*. Each rule R_i has the following structure:

$$x_1, \dots, x_n : a_1, \dots, a_m; F_i; \quad (4.1)$$

The condition and action parts are separated by a colon, and are formed by a set of input and output variables, $x_i \in \{0, 1\}$ and $a_j \in \{0, 1\}$ with $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$, respectively. The parameter F_i represents the strength of the rule. Rules define relationships between input and output variables by letting them take some specific values. These variables, in the condition and action parts, can also take the symbol #. The symbol # has different meanings depending on where it appears. A # symbol in the condition part means that that place could take the value either 0 or 1. A # symbol in the action part of a rule has two meanings. The first is when the rule is going to post a message that determines an action (an action message), in this case the value of that position does not matter. The second is when the rule is going to post a message for rules (a rule message), in this case it takes the value of the same position in the message that matched the rule. In this way, the rule $00\#1 : 01\#\#$ is matched by the messages 0001 and 0011. That rule would propose the action messages 01#\# or 01#\# and the rule messages 0101 or 0111 depending of which message 0001 or 0011 matched the rule.

- **Performance Component.** It determines the way in which the LCS decides what to do. Each time step, the perceived state \vec{x} of the environment is transformed into rule messages that are posted in the current list of messages. Then, these messages are used to match rules in the population of rules $[P]$. In this way, a matching set $[M]$ that includes all the matched rules is formed. At this point, which rule from $[M]$ is going to post its action message or rule message on a new message list is determined. To do this selection, first each rule $R_i \in [M]$ gives a bid proportional to its strength, $B_i = \beta F_i$ with $\beta \in [0, 1]$, and then, a rule from $[M]$ is selected with probability proportional to its bid B_i . The message of the selected rule \vec{a} is posted in the new message list. The current message list is replaced by the new message list. Each time step, the message list is checked to detect action messages. If there are actions messages on the message list, then the actions proposed are performed. This procedure is repeated while the agent is acting on the environment. Figure 4.1 depicts how the performance component works. It can be seen how an action message is differentiated from a rule message by the first bit. That bit means 1 for action messages and 0 for rule messages. All

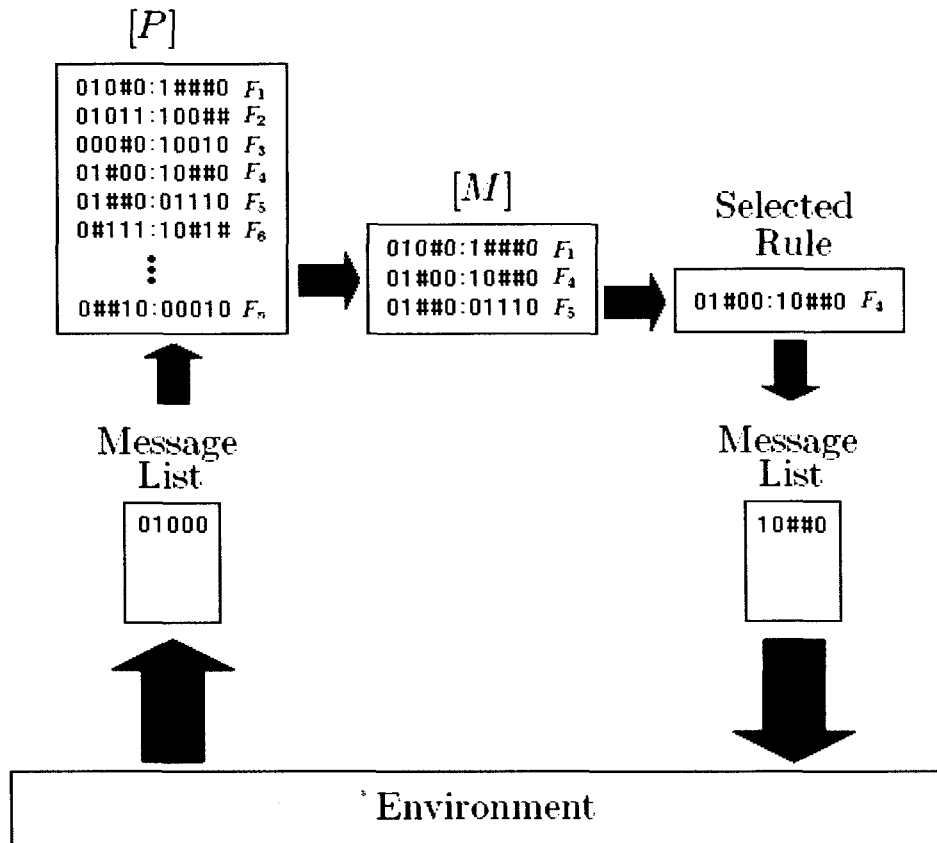


Figure 4.1: Performance Component of the Holland's LCS

of the rules in $[P]$ also have 0 in the first position, because they are referring to rule messages. This example uses tags to differentiate the type of messages. The use of tags is common but not indispensable.

- **Learning Component.** This component is used to learn the best rules for the task in the corresponding environment. Holland devised a learning mechanism based in the way that commerce works. This mechanism was called the Bucket-Brigade Algorithm (BBA). In BBA when a rule $R_i \in [M]$ is selected, it has to pay its bid B_i to the message with which it matched, and it receives a payment when the posted message receives payment. Its message can receive payment from another rule that match it or from the environment. So each selected rule modifies its strength in the following way:

$$F_i \leftarrow F_i - B_i + R, \quad (4.2)$$

where R is the reward from the environment.

- **Discovery Component.** This component is the part of the system that creates and deletes rules. It decides which rules $R_i \in [P]$ are good and which ones are bad based on their strengths F_i . So the one that has the higher F_i is the best one. The worst rule is deleted when a new one is created. It uses a GA to create new rules. It selects two rules from $[P]$ with probability proportional to their F_i . Then it applies crossover with probability χ and mutation with probability μ . The rule created is placed in $[P]$. This mechanism creates a new rule on intervals of time. Therefore, this component allows the agent to determine from the environment which rules are the best based on what they have learned by the learning component.

This LCS can represent generalizations because of the symbol $\#$. This means that a rule can associate an action to many states. This LCS could also chain rules. This can be seen through the mechanism of the messages. A rule can post a rule message without performing an action. In this manner, one rule can activate another rule and, that other rule another one, and so on, as shown in Fig. 4.2. In this manner, the system has the ability of planning. One more thing this system could do would be the creation of default-exception rules. In this case, the creation of a pair of rules, where one is a general rule that works well over most of its domain except for a small subset of it in which the other rule works better, would be possible. For example, let us have a set of possible states of the environment that are associated with their correct actions as follows:

$$\begin{array}{ll} \vec{x} & \vec{a} \\ 1000, & 00100; \\ 10001, & 10010; \\ 10100, & 00100; \\ 10101, & 00100. \end{array} \quad (4.3)$$

There are three states out of four that have the same action. If all the states had the same action 00100 it would be easy to represent them by only one rule R_1 10#0#:00100. But, there is only one state where R_1 fails. In that state, the next rule R_2 10100:10010

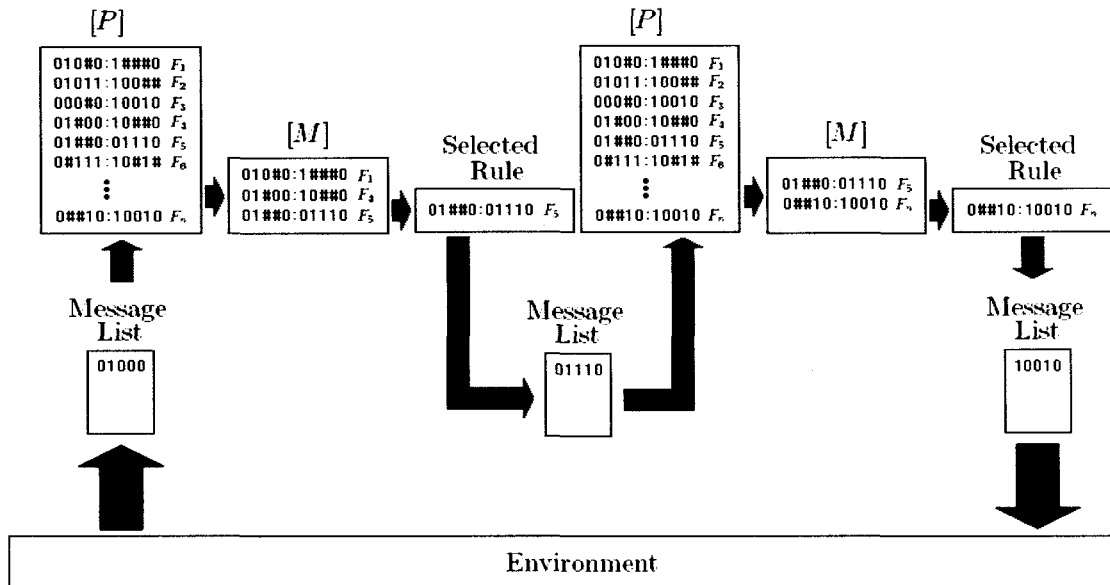


Figure 4.2: Rule chaining

could be created. If R_2 were only used in that state and R_1 in the rest three states there would be a pair of rules working together doing their best in those four states. The rule R_1 is called the *default* and the rule R_2 is called the *exception*. In this way, the number of rules needed to represent the environment would be reduced. However, the LCS could not exhibit neither chained rules nor general-exception rules.

In general, the LCS used a scheme of paying and receiving bids as learning. The amount of these received bids was accumulated in a parameter called strength. The GA evolved classifiers based on strength. This LCS by its nature can just be applied to discrete problems.

4.2 LCSs with Fitness Based on Accuracy

An important approach made on LCSs is the idea of introducing accuracy. It originated the LCS that was called XCS [33]. It can deal with discrete problems with great success. It was tested on discrete non-Markov problems [17] too. Many attempts have been made to modify XCS to deal with real input variables [26, 35, 8]. These efforts carried on to the design of the XCSF [34] that was proved in continuous spaces. The following two sections show how XCS and XCSF work.

4.2.1 XCS

Wilson simplified Holland's LCS [13] framework avoiding the idea of using a message list. He designed a LCS that can act on the environment immediately after perceiving it. So, in XCS [33, 9] rules do not post messages for other rules. This means the system perceives and acts over and over again. Wilson used Q-learning in XCS [24, 27] instead

of Holland's Bucket-Brigade Algorithm so the system was able to learn by rewards. He introduced the accuracy of the expected reward instead of strength to determine the quality of rules. This quality determined which rules were used by the GA to create new ones and which rules were deleted. XCS has the next components:

- **Population of Rules.** It has a set of rules $[P]$. This set is called *population*. Each rule R_i has the following structure:

$$\begin{aligned} x_1, \dots, x_n : a_1, \dots, a_m; & p_i; \\ & e_i; \\ & F_i; \end{aligned} \quad (4.4)$$

where the condition and action parts are separated by a colon and are formed by a set of input and output variables, $x_i \in \{0, 1\}$ and $a_j \in \{0, 1\}$ with $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$, respectively. There are three parameters that are the prediction p_i , the error of the prediction e_i , and the accuracy of the prediction called fitness F_i . Rules define relationships between input and output variables by letting them take some specific values. In the condition part of a rule, a variable can also take the symbol $\#$. $\#$ means that the corresponding variable can take the value either 0 or 1, in other words, it does not matter which value it takes.

- **Performance Component.** It determines how XCS works when the agent is acting in the environment. First, it takes a perception that is a discrete vector \vec{x} . This perception is used to get the rules that match it, which form the matching set $[M]$. From those rules $R_i \in [M]$ the prediction array \vec{P} is calculated as follows:

$$\vec{P} = [P(\vec{a}_1), P(\vec{a}_2), \dots, P(\vec{a}_{2^m})], \quad (4.5)$$

with

$$P(\vec{a}_k) = \frac{\sum_{R_l \in [M]_k} F_l p_l}{\sum_{R_{l'} \in [M]_k} F_{l'}}, \quad (4.6)$$

and $k = 1, 2, \dots, 2^m$. \vec{a}_k is the k -th action vector. There are 2^m possible action vectors $\vec{a} = (a_1, a_2, \dots, a_m)$ since the variables are binaries. p_l , F_l , R_l and $[M]_k$ represent the prediction of the l -th rule, the fitness of the l -th rule, the l -th rule and a subset of rules in $[M]$ that contain the k -th action vector \vec{a}_k , respectively. With \vec{P} , there are two ways of selecting the action that it is going to be done. The first one is when the action selected \vec{a}_E has the maximum value in \vec{P} :

$$\vec{a}_E = \arg \max_{\vec{a}} [P(\vec{a}_1), P(\vec{a}_2), \dots, P(\vec{a}_{2^m})], \quad (4.7)$$

and the other one is when the action \vec{a}_R is selected randomly. These two ways of selecting actions have to be combined with probabilities P_E of applying the first one and $P_R = (1 - P_E)$ of applying the second one. The former procedure is known as *exploitation* and the latter as *exploration*. Then an action set $[A]$ is formed with the rules that propose the selected action, either \vec{a}_E or \vec{a}_R , and finally the action is performed in the environment. This procedure is repeated over and over again and is depicted in Fig. 4.3.

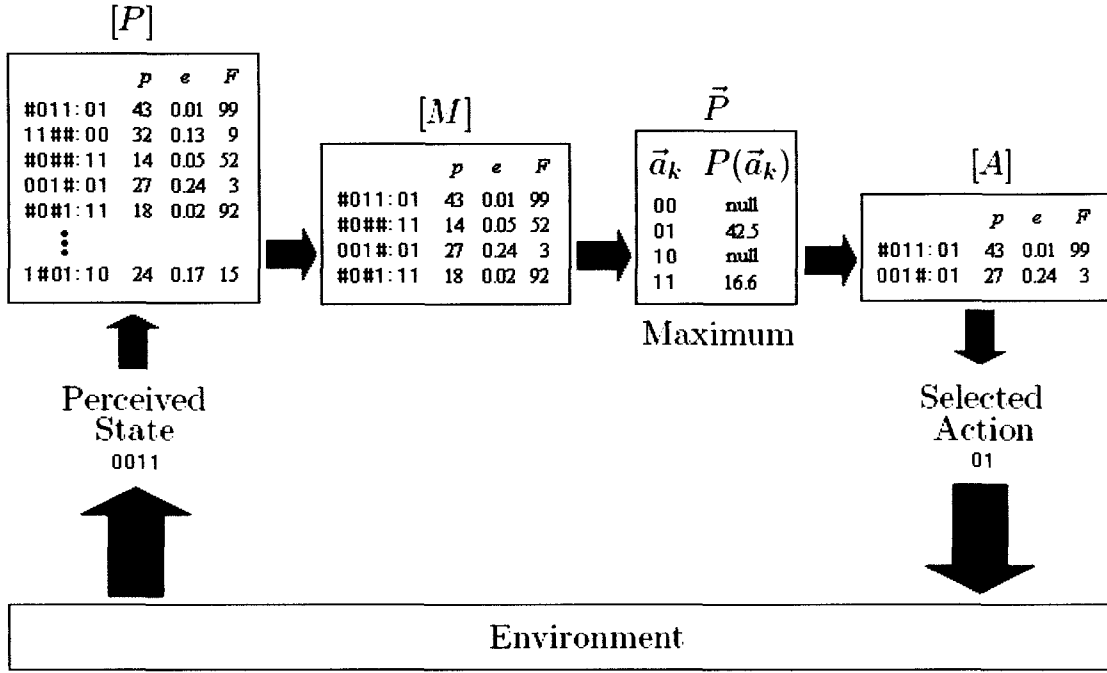


Figure 4.3: Performance Component of the XCS

- **Learning Component.** After each action given by the performance component, the agent receives a reward. This reward is a real function $R(\vec{x})$ over all the possible perceived states \vec{x} . This function is a requirement for Q-learning. So the prediction p_l of each rule represents the Q-values. The states (\vec{x}, \vec{a}) are represented by the condition and the action of the rules. In this way, the Q-function is part of the rules. Each rule is able to represent, not only one state (\vec{x}, \vec{a}) , but several states that have the same action and the same Q-value. This component updates the parameter p_l on the rules from the action set that proposed the previous action $[A]_{t-1}$ by:

$$p_l \leftarrow p_l + \beta \left[\left(R(\vec{x}_{t-1}) + \gamma \max_{\vec{a}} \vec{P}_t \right) - p_l \right], \quad (4.8)$$

where $\beta \in [0, 1]$, $\gamma \in [0, 1]$ and p_l , \vec{x}_{t-1} and \vec{P}_t are the prediction of the l -th rule $R_l \in [A]_{t-1}$, the perceived state \vec{x} in the time $t-1$ and the vector of prediction at time t , respectively. Eq. 4.8 is similar to Eq. 3.12 in Q-learning. XCS, as it was said before, incorporated not only the Q-learning mechanism, but also two more parameters e_l and F_l in each rule that determine the convergence of the Q-values and the fitness, respectively. The convergence of the Q-value is measured through the error in the prediction of p_l and the fitness is determined by an accuracy measure. The more accurate a rule is, the more fitness it has. A rule is more accurate if it has a smaller error. So these parameters are also updated in the

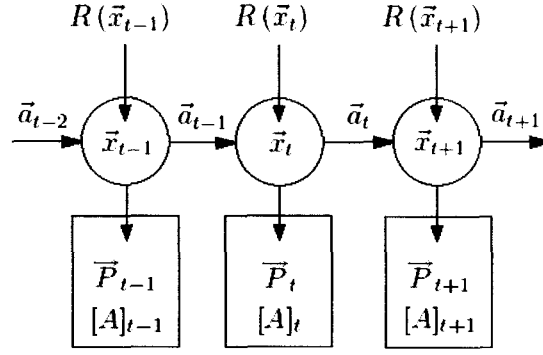


Figure 4.4: Learning Component of the XCS

rules $R_l \in [A]_{t-1}$ as follows. For error e_l is:

$$e_l \leftarrow e_l + \beta \left[\left| \left(R(\vec{x}_{t-1}) + \gamma \max_{\vec{a}} \vec{P}_t \right) - p_l \right| - e_l \right], \quad (4.9)$$

where e_l is the error of the l -th rule R_l and the other factors are the same ones as in update 4.8. And for the fitness F_l is:

$$F_l \leftarrow F_l + \beta (\kappa_l - F_l), \quad (4.10)$$

with

$$\kappa_l = \frac{k_l}{\sum_{R_{l'} \in [A]_{t-1}} k_{l'}}, \quad (4.11)$$

and

$$k_l = \begin{cases} e^{\left(\frac{e_l - e_0}{e_0}\right) \ln \alpha}, & \text{if } e_l > e_0; \\ 1, & \text{otherwise,} \end{cases} \quad (4.12)$$

where F_l , κ_l and k_l are the fitness of the l -th rule R_l , the relative accuracy of the l -th rule R_l , and the accuracy of the l -th rule R_l . The other factors are the same ones as in update 4.8. In Fig. 4.4, the relationships over time among the factors used in adjusting the parameters p_l , e_l and F_l of each rule is shown. It represents a sequence of actions done by the agent. In each state on the time it receives reward from the environment, it calculates the prediction vector \vec{P} , decides what action to do, gets the action set $[A]$, and changes the parameters p_l , e_l , and F_l on the rules from $[A]$ in the previous time step.

- **Discovery Component.** XCS has two ways of generating new rules. The first one, called *covering*, creates random rules when $[M]$ is empty. These rules have to match the perceive state. The second one uses a GA on the rules from the previous time step $[A]$. In this way, two rules are selected with probability proportionally to their fitness F_l . They are crossed-over and mutated with probability χ and μ , respectively. XCS also deletes rules in such a way that it maintains all possible $[A]$ s with the same number of rules and with their fitness over the average fitness

of the population $[P]$. The probability of deleting the l -th rule P_D^l is calculated as:

$$P_D^l = \begin{cases} \left[\frac{\bar{F}}{F_l} \right] E_l, & \text{if } F_l < \delta \bar{F} \text{ with } \bar{F} = \frac{\sum_{R_{l'} \in [P]} F_{l'}}{N}; \\ E_l, & \text{otherwise,} \end{cases} \quad (4.13)$$

where R_l , \bar{F} , F_l , E_l and N are the l -th rule $R_l \in [P]$, the average fitness of the rules on $[P]$, the fitness of the l -th rule R_l , an estimation of the sizes of $[A]$ s where R_l have been, and the number of rules in the population $[P]$. E_l is updated in the rules $R_l \in [A]$ each time $[A]$ is created as:

$$E_l \leftarrow E_l + \beta [N_{[A]} - E_l], \quad (4.14)$$

where $N_{[A]}$ is the number of rules of $[A]$.

The accuracy concept measures the convergence of the Q-values of rules. Thus, the fitness is higher when the classifier is accurate. Then, the GA evolves classifiers based on this accuracy. The used learning algorithm is Q-learning.

XCS can find rules that exhibit a kind of generalization called *maximally generalization*. In this type of generalization, a rule assigns one correct action to as many states of the perceived environment as possible with the constraint that all of the possible pairs (\vec{x}, \vec{a}) , represented by the rule, share approximately the same prediction value p_l . It means that it is not always possible to assign a rule to all possible states that share the same action and the same value p_l .

XCS worked well in simple environments like the n -multiplexer and woods1. The n -multiplexer consist of binary strings as states. The string is divided into two subsets A and B . A combination of 0s and 1s in subset A represent one binary element of the subset B . Therefore, the actions of the problem are to place in the output the value of the bit in B that was selected by the set A . For example: in the string 100110, $A = 10$ and $B = 0110$, thus if combination 01 represents the third bit from left to right of B then to place 1 as the output is the action. The other problem, woods1 was a navigation problem in a periodic two-dimensional grid space of 5×5 . The system had to reach the goal that was in the corner of the objects. There was a big obstacle which the system could not pass. The actions were go up, down, left and right. This problem is shown in Fig. 4.5. The obstacles are in dark gray and the goal in gray. These problems are discrete.

4.2.2 XCSF

XCS [33] was modified to deal with real input variables. The main changes are in the type of input variables from discrete to continuous, the domain of the output variables, the way of predicting p_l that is done as a linear combination of the input variables, and the way of applying the GA that is modified to work with continuous values. XCSF [34] has the same components as XCS but with its respective changes. The description of these changes is given next.

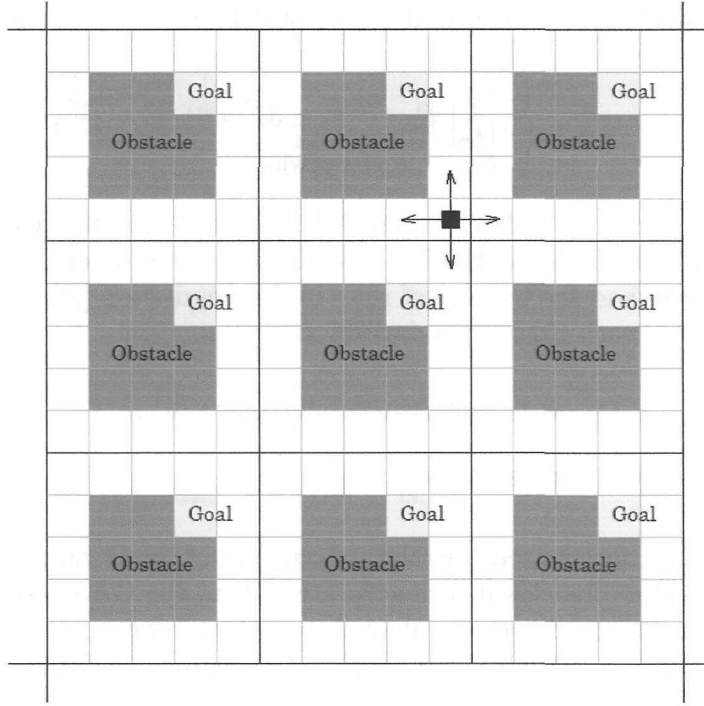


Figure 4.5: Woods1

- **Population of Rules.** It has a fix population of rules $[P]$. Each rule R_l has the following structure:

$$\begin{aligned}
 x_0^l, [\min_{x_1}^l, \max_{x_1}^l], \dots, [\min_{x_n}^l, \max_{x_n}^l] : a; \quad \vec{w}_l = (w_0^l, w_1^l, \dots, w_n^l); \\
 e_l; \\
 F_l; \\
 n_l;
 \end{aligned} \tag{4.15}$$

where x_0^l , $[\min_{x_i}^l, \max_{x_i}^l]$, $a \in \{a_1, a_2, \dots, a_m\}$, \vec{w}_l , e_l , F_l and n_l are a constant, the lower and upper limits of an interval for the i -th continuous input variable $x_i \in [x_i^{\min}, x_i^{\max}]$, a discrete variable for an action, the weight vector that has a constant for each input variable including the constant x_0^l , the error, the fitness and the numerosity that represents the number of copies in the population of the l -th rule, respectively. The colon separates the condition from the action.

- **Performance Component.** It determines how XCSF works when the agent is acting in the environment. First, it takes a perception that is a real vector \vec{x} . This perception is used to obtain the matched rules $[M]$. A rule is matched when each component of \vec{x} are in the corresponding intervals of its condition. From those rules in $[M]$, the prediction array \vec{P} is calculated as follows:

$$\vec{P} = [P(a_1), P(a_2), \dots, P(a_m)], \tag{4.16}$$

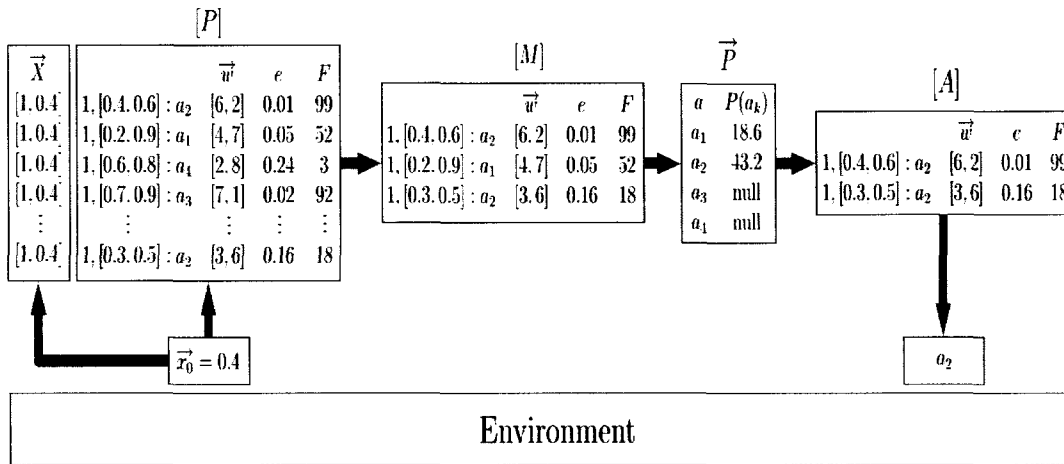


Figure 4.6: Performance component of the XCSF

with

$$P(a_k) = \frac{\sum_{R_l \in [M]_k} F_l p_l}{\sum_{R_{l'} \in [M]_k} F_{l'}}, \quad (4.17)$$

$$\text{and } p_l = \vec{w}_l \cdot \vec{x}^t, \text{ where } \vec{x}^t = (x_0^t, \vec{x}). \quad (4.18)$$

p_l represents the prediction of R_l and $[M]_k$ is the set of rules in $[M]$ that have the k -th action a_k . With \vec{P} there are two ways to select the action that is going to be done, by either exploitation or exploration. The first is when the action a_E with the maximum value in \vec{P} is selected as

$$a_E = \arg \max_a [P(a_1), P(a_2), \dots, P(a_m)], \quad (4.19)$$

and the second is when the action a_R is selected randomly. These two ways of selecting actions are combined with probabilities P_E of applying the first one and $P_R = (1 - P_E)$ of applying the second one. Then an action set $[A]$ is formed with the rules that propose the selected action either a_E or a_R and finally the action is performed in the environment. This procedure is shown in Fig. 4.6 and is repeated over and over again.

- **Learning Component.** In this component, the weight vector \vec{w}_l , the error e_l of the prediction and the fitness F_l of the rules $R_l \in [A]_{t-1}$ are updated as follows:

$$\vec{w}_l \leftarrow \vec{w}_l + \frac{\eta}{|\vec{x}_{t-1}^t|^2} \left[\left(R(\vec{x}_{t-1}) + \gamma \max_a \vec{P}_t \right) - p_l \right] \vec{x}_{t-1}^t, \quad (4.20)$$

$$e_l \leftarrow e_l + \beta \left[\left| \left(R(\vec{x}_{t-1}) + \gamma \max_a \vec{P}_t \right) - p_l \right| - e_l \right], \quad (4.21)$$

$$F_l \leftarrow F_l + \beta (\kappa_l - F_l), \quad (4.22)$$

with

$$\kappa_l = \frac{k_l n_l}{\sum_{R_{l'} \in [A]_{t-1}} k_{l'} n_{l'}}, \quad (4.23)$$

and

$$k_l = \begin{cases} \left(\frac{e_l}{e_0}\right)^\alpha, & \text{if } e_l > e_0; \\ 1, & \text{otherwise,} \end{cases} \quad (4.24)$$

where $\eta \in [0, 1]$, $\gamma \in [0, 1]$, $\beta \in [0, 1]$, α is a constant, e_0 is the error threshold, \vec{x}_{t-1}^l is \vec{x}^l at time $(t-1)$, \vec{x}_{t-1} is \vec{x} at time $(t-1)$, \vec{P}_t is \vec{P} at time t , κ_l is the relative accuracy of the l -th rule, k_l is the accuracy of the l -th rule and n_l is the number of copies of the l -th rule in $[P]$.

- **Discovery Component.** The creation and deletion of rules is similar to the one used in XCS [33]. It uses a GA and a covering mechanism to create new rules that are inserted in $[P]$. The GA takes two rules based on their fitness and applies two-point crossover and mutation with probabilities χ and μ , respectively. In the GA, mutation is applied over the condition parts of the rules by adding a real number randomly on the interval $[-\mu_0, \mu_0]$ where μ_0 is a constant; when mutation is applied over the action parts it takes an action randomly. The offspring inherits \vec{w}_l from their parents. The covering mechanism creates rules with $\vec{w}_l = \vec{0}$, action a at random, and the limits of the intervals of the variables x_i from the condition parts of the rules are set randomly in $[x_i^{\min}, x_i^{\max}]$. The way of deleting rules is the same as in XCS [33].

First, XCSF was designed to be capable of learning continuous functions. It had one action that was not used. The functions were represented by the Q-values.

The main contribution of XCSF was the calculus of the prediction through a dot product between the input vector and the weight vector. This made classifiers approximate the Q-function by hyper-planes. In general, XCSF had a set of discrete actions. Therefore, the classifier associated a continuous set of input vectors with one action.

XCSF was applied to many navigation tasks [15] that were: the discrete and continuous linear corridor problem and the 2D discrete and continuous gridWorld problem. These problems are shown in Fig. 4.7. In these learning tasks, XCSF perceived continuous inputs and chose an action from a set of discrete actions.

4.3 Fuzzy LCSs

The incorporation of FL [32] to LCSs has been sought in the hope of being able to deal with continuous spaces because of the many problems of the real life that can be represented in that way. There are two different and important approaches to FCSs that are worth describing. The former is due to the FCS by Valenzuela [30, 31] and the FCS by Parodi and Bonelli [20]. They learn continuous functions with fixed and unfixed fuzzy sets, respectively. The latter is due to Bonarini [3]. He uses a dilemma

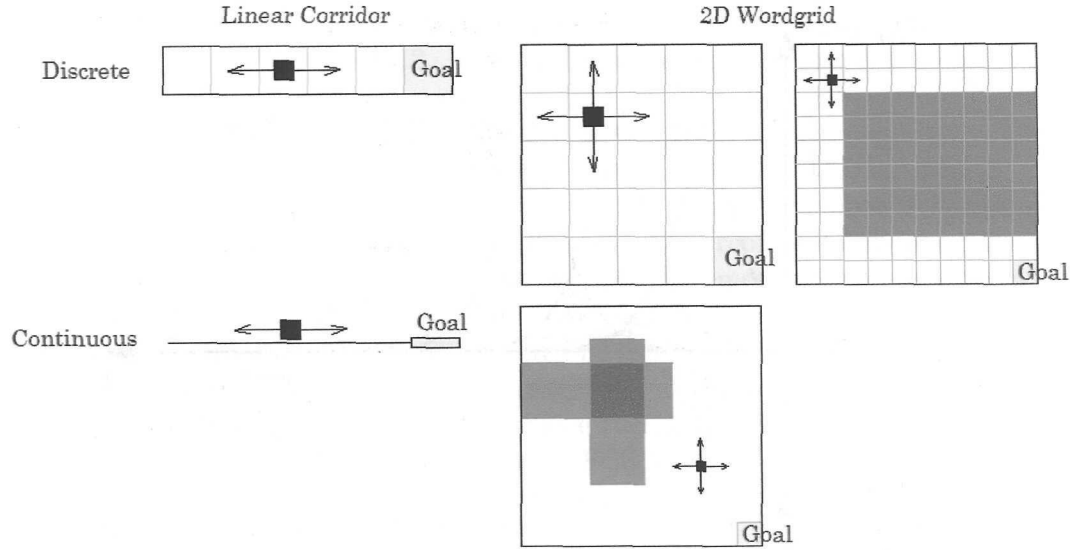


Figure 4.7: The linear corridor problem and the 2D GridWorld problem

of competition and cooperation among rules. The substantial difference between the traditional approach and the fuzzy approach is that in the former the rules compete and in the latter they cooperate. But in Bonarini's scheme, rules compete and cooperate at the same time. Bonarini's scheme has been used to learn continuous action functions using reinforcement learning.

4.3.1 Valenzuela's FCS

This LCS [29, 30, 31] uses fuzzy rules that cooperate based on their membership values. Therefore, a rule with a big membership value has a larger influence on the output of the system and vice versa as it is in FL [32]. Rules received payment from the environment using a modification of the BBA (Eq. 4.2) that determines the amount of payment they receive based on their membership values. The FCS is compound by a set of linguistic variables, a list of minimal messages, a population of fuzzy rules, a performance component, a learning component, and a discovery component.

- **Linguistic Variables.** It has a set of linguistic variables $(\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n, \mathcal{A})$ that match with the corresponding continuous variables $(x_1, x_2, \dots, x_n, a)$ that represent the input and the output variables. In each variable a set of linguistic values is defined with membership function given by:

$$\mu_{X_{ir}}(x_i) = \frac{4e^{-\frac{(x_i - h_{ir})}{\sigma_i}}}{\left(1 + e^{-\frac{(x_i - h_{ir})}{\sigma_i}}\right)^2}, \quad (4.25)$$

$$\mu_{A_s}(a) = \frac{4e^{-\frac{(a - h_s)}{\sigma}}}{\left(1 + e^{-\frac{(a - h_s)}{\sigma}}\right)^2}, \quad (4.26)$$

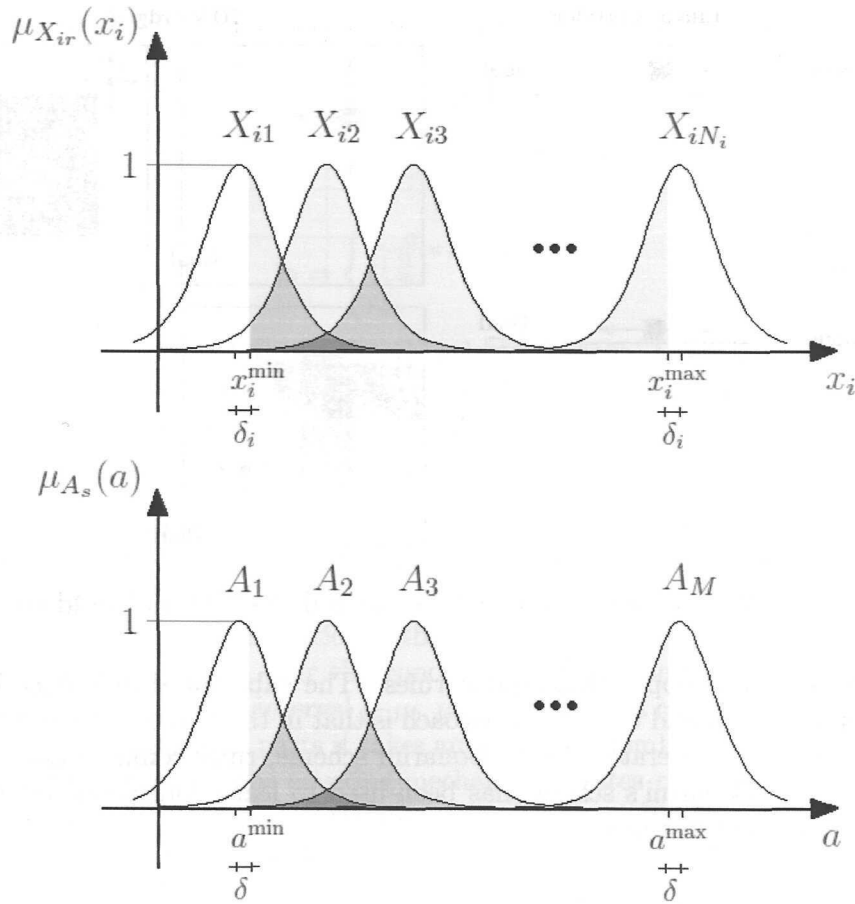


Figure 4.8: Membership functions of a linguistic variable in Valenzuela's FCS

where $\mu_{X_{ir}}(x_i)$ (or $\mu_{A_s}(a)$) is the membership function for r -th fuzzy value of the i -th linguistic variable \mathcal{X}_i (or for s -th fuzzy value of the linguistic variable \mathcal{A}) that approximate a normal function with parameter σ_i (or σ). The parameter h_{ir} (or h_s) is define by:

$$h_{ir} = (r - 1) \frac{(x_i^{\max} + \delta_i) - (x_i^{\min} - \delta_i)}{N_i - 1} + (x_i^{\min} - \delta_i), \quad (4.27)$$

$$h_s = (s - 1) \frac{(a^{\max} + \delta) - (a^{\min} - \delta)}{M - 1} + (a^{\min} - \delta), \quad (4.28)$$

where $[x_i^{\min}, x_i^{\max}]$, N_i and δ_i (or $[a^{\min}, a^{\max}]$, M and δ) are the range of the x_i (or a), the number of the fuzzy values on \mathcal{X}_i (or \mathcal{A}) and a parameter that controls the center of the first and the last membership functions of the fuzzy values of \mathcal{X}_i (or \mathcal{A}), respectively. Figure 4.8 shows the membership functions of the linguistic variables.

- **List of Minimal Messages.** In this FCS messages exist. There is a message for each fuzzy value. It is to say that, the r -th (or s -th) message \mathcal{M}_{ir} (or \mathcal{M}_s)

represents the r -th (or s -th) fuzzy value from the linguistic variable \mathcal{X}_i (or \mathcal{A}) with $r \in \{1, 2, \dots, N_i\}$ (or $s \in \{1, 2, \dots, M\}$). The structure of a minimal message is:

$$\begin{array}{ll} \text{For } \mathcal{X}_i & \text{For } \mathcal{A} \\ i, 000\dots00100, \mu^{ir}; & 000\dots00010, \mu^s; \end{array} \quad (4.29)$$

that contains the number i of variable \mathcal{X}_i (or without number if it belongs to variable \mathcal{A}), a sequence of N_i (or M) 0s and 1s that represent only one fuzzy value in \mathcal{X}_i (or \mathcal{A}) that means there is only one 1 in each message, and an activity level μ^{ir} (or μ^s). So a 1 in the r -th (or s -th) position means that the r -th (or s -th) fuzzy value from \mathcal{X}_i (or \mathcal{A}) is represented by this message and a 0 determines fuzzy values that are not taken by this message. For example 4:0100:0.2 means the 2-th fuzzy values of \mathcal{X}_4 with $\mu^{42} = 0.2$. Since the messages represent only one fuzzy value, they are called *minimal messages*. FCS has a list of these minimal messages that represent a perception or an action on the environment.

- **Population of Rules.** FCS has a fixed population of N rules $[P]$. Each rule R_i has a condition, an action, an activity level μ_i and strength F_i as follows:

$$\text{IF } [\mathcal{X}_1 = X_1^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A} = A^l]; \quad \begin{array}{l} \mu_i; \\ F_i; \end{array} \quad (4.30)$$

where X_i^l (or A^l) is a fuzzy disjunction of a subset of fuzzy values S_i^l (or S^l) of the fuzzy values of linguistic variable \mathcal{X}_i (or \mathcal{A}). It means $S_i^l \subset \{X_{i1}, X_{i2}, \dots, X_{iN_i}\}$ (or $S^l \subset \{A_1, A_2, \dots, A_M\}$). These rules are coded in the next way:

$$1100, \dots, 1010 : 0110; \quad \begin{array}{l} \mu_i; \\ F_i; \end{array} \quad (4.31)$$

where commas represent the AND connectors and the sequence of 0s and 1s between AND connectors represent the fuzzy values X_i^l (or A^l) using a similar codification than in the minimal messages but in this case there can be more than one 1 since X_i^l (or A^l) is the fuzzy disjunction of fuzzy values in S_i^l (or S^l). This characteristic represents the ability of generalizing. A colon separates the condition from the action.

- **Performance Component.** This component determines the way the FCS acts. First, it fuzzifies the input real vector \vec{x}_0 into a set of input minimal messages. Then, the inference machine produces an output fuzzy fact A' . And finally, it defuzzifies the output fuzzy fact A' to produce the real output a_0 and the set of output minimal messages that are needed for the learning.

- **Fuzzification.** It represents the input real vector $\vec{x}_0 = (x_1^0, x_2^0, \dots, x_n^0)$ into the set of minimal messages \mathcal{M}_{ir} . To achieve this, the activity level μ^{ir} of the r -th minimal message \mathcal{M}_{ir} is obtained by

$$\mu^{ir} = \mu_{X_{ir}}(x_i^0), \quad (4.32)$$

Then, for all the minimal messages that have their μ^{ir} below a minimal value, their μ^{ir} are set to zero.

- **Inference Machine.** Each minimal message \mathcal{M}_{ir} with $\mu^{ir} \neq 0$ in the message list is used to match rules. Each message \mathcal{M}_{ir} matches a rule when its fuzzy value $X_{ir} \in S_i^l$. A rule is satisfied when all of its input variables have matched at least one message. A satisfied rules form the match set $[M]$. The fuzzy fact A' is obtained by the combination of all fuzzy facts A'_i of rules $R_i \in [M]$ as

$$\mu_{A'}(a) = \sum_{R_i \in [M]} \mu_{A'_i}(a). \quad (4.33)$$

The fuzzy fact A'_i of each rule $R_i \in [M]$ is form as

$$\mu_{A'_i}(a) = \left[\frac{B_i}{|S^l|} \right] [\tilde{a}_1^l \mu_{A_1}(a) + \tilde{a}_2^l \mu_{A_2}(a) + \dots + \tilde{a}_M^l \mu_{A_M}(a)] \quad (4.34)$$

where $|S^l|$ means the cardinality of S^l , $\tilde{a}_s^l \in \{0, 1\}$ determines the fuzzy value A_s that is in the rule R_i as

$$\tilde{a}_s^l = \begin{cases} 1, & \text{if } A_s \in S^l; \\ 0, & \text{otherwise,} \end{cases} \quad (4.35)$$

and B_i means the bid of the rule $R_i \in [M]$ calculated as

$$B_i = \beta \mu_i F_i, \quad (4.36)$$

where μ_i is obtained as:

$$\mu_i = \min [\mu_1^l, \mu_2^l, \dots, \mu_n^l], \quad (4.37)$$

where

$$\mu_i^l = \max [\tilde{x}_{i1}^l \mu^{i1}, \tilde{x}_{i2}^l \mu^{i2}, \dots, \tilde{x}_{iN_i}^l \mu^{iN_i}], \quad (4.38)$$

with

$$\tilde{x}_{ir}^l = \begin{cases} 1, & \text{if } (X_{ir} \in S_i^l) \wedge (\mu^{i1} \neq 0); \\ 0, & \text{otherwise.} \end{cases} \quad (4.39)$$

- **Defuzzification.** The output fuzzy fact A' is defuzzified by the center of gravity (Eq. 2.41) to produce the real output $a = a_0$. This output is represented by the set of minimal messages \mathcal{M}_s with μ^s as

$$\mu^s = \sum_{R_i \in [M]} \tilde{a}_s^l \left[\frac{B_i}{|S^l|} \right]. \quad (4.40)$$

These minimal messages are needed because the payoff is done through them.

Figure 4.9 shows an example with a few classifiers that follows the procedure described.

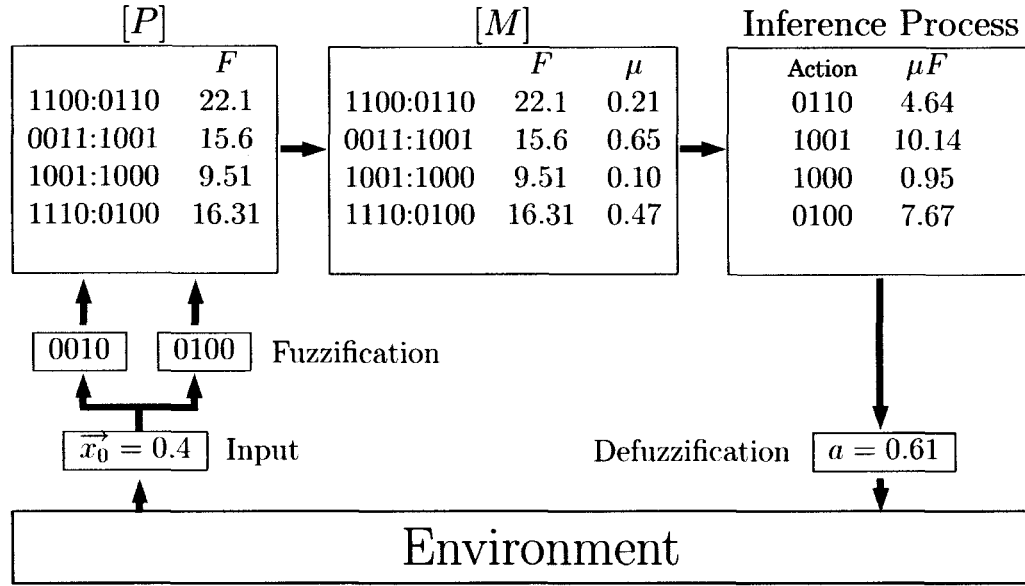


Figure 4.9: Diagram of the performance component of the Valenzuela's FCS.

- **Learning Component.** When FCS is learning, in each step of the performance component B_l (Eq. 4.36) is calculated as:

$$B_l = \beta \mu_l F_l [1 + N(\sigma_{\text{noise}}^2)] \quad (4.41)$$

where $N(\sigma_{\text{noise}}^2)$ represents a randomly number with a Gaussian probability density with standard deviation σ_{noise} . Matched rules have to pay their bids to the messages with which they matched. This payment is divided in equal parts into the input messages. The environment gives reward to output messages. This reward is divided into the output messages proportionally to their activity levels. Then each output message pays its reward to the rules that post it proportionally to their contributions. So matched rules receive payment from the messages they post.

- **Discovery Component.** A GA is used to create new rules. Two rules are selected according to their F_l , they are crossed-over and mutated with probabilities χ and, μ respectively and finally one of the offspring is inserted in $[P]$ substituting the worst rule. The GA is applied after θ_{GA} steps of the performance component.

The FCS uses fixed fuzzy sets. Classifiers are represented by fuzzy rules. The competition is replaced by cooperation because that is how rules are combined to produce outputs in Fuzzy Logic. The FCS is actually a fuzzy system. For learning, it uses a similar scheme to the Holland's LCS. It is based on receiving and paying bids. The bids are accumulated. Rules are evolved by a GA based on strength. The FCS was applied to learn continuous functions. This problem is single step.

4.3.2 Parodi and Bonelli's FCS

This FCS [20] is similar to Valenzuela's FCS [30, 31] but it does not have a message list and its fuzzy values can change during the learning. This FCS has a set of linguistic variables, a population of rules, a performance component, a learning component, and a discovery component.

- **Linguistic Variables.** It has a set of linguistic variables $(\mathcal{X}_1, \dots, \mathcal{X}_n, \mathcal{A})$ that match with the corresponding continuous variables (x_1, \dots, x_n, a) and represent the input and the output variables. In each variable, there are N linguistic values with triangular membership function defined as

$$\mu_{X_i^l}(x_i) = \begin{cases} \left[\frac{2}{w_i^l} \right] x_i + \left[1 - \frac{2c_i^l}{w_i^l} \right], & \text{if } \left[c_i^l - \frac{w_i^l}{2} \right] \leq x_i < c_i^l; \\ - \left[\frac{2}{w_i^l} \right] x_i + \left[1 + \frac{2c_i^l}{w_i^l} \right], & \text{if } c_i^l \leq x_i < \left[c_i^l + \frac{w_i^l}{2} \right]; \\ 0, & \text{otherwise,} \end{cases} \quad (4.42)$$

$$\mu_{A^l}(a) = \begin{cases} \left[\frac{2}{w^l} \right] a + \left[1 - \frac{2c^l}{w^l} \right], & \text{if } \left[c^l - \frac{w^l}{2} \right] \leq a < c^l; \\ - \left[\frac{2}{w^l} \right] a + \left[1 + \frac{2c^l}{w^l} \right], & \text{if } c^l \leq a < \left[c^l + \frac{w^l}{2} \right]; \\ 0, & \text{otherwise,} \end{cases} \quad (4.43)$$

where $\mu_{X_i^l}(x_i)$, c_i^l and w_i^l (or $\mu_{A^l}(a)$, c^l and w^l) with $l \in \{1, 2, \dots, N\}$, are the membership function of the fuzzy value X_i^l (or A^l) of the variable \mathcal{X}_i (or \mathcal{A}), the center of the $\mu_{X_i^l}(x_i)$ (or $\mu_{A^l}(a)$) and the wide of $\mu_{X_i^l}(x_i)$ (or $\mu_{A^l}(a)$) respectively.

- **Population of Rules.** This FCS has a fixed population of N rules $[P]$. Each rule $R_l \in [P]$ has a condition, an action, an activity level μ_l and strength F_l as it follows:

$$\text{IF } [\mathcal{X}_1 = X_1^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A} = A^l]; \quad \begin{matrix} \mu_l; \\ F_l; \end{matrix} \quad (4.44)$$

where X_i^l , A^l , μ_l and F_l denote the l -th fuzzy value associated to \mathcal{X}_i in the l -th rule, the l -th fuzzy value associated to \mathcal{A} in the l -th rule, the activity level of the l -th rule, and the strength of the l -th rule. These rules are coded in the next way:

$$[c_1^l, w_1^l], \dots, [c_n^l, w_n^l] : [c^l, w^l]; \quad \begin{matrix} \mu_l; \\ F_l; \end{matrix} \quad (4.45)$$

where $[c_i^l, w_i^l]$ (or $[c^l, w^l]$) represents the center and the wide of $\mu_{X_i^l}(x_i)$ (or $\mu_{A^l}(a)$) and the commas the AND connectors.

- **Performance Component.** This component determines the way the FCS acts. First, it fuzzifies the input real vector \vec{x}_0 into a set of membership values. Then, the Inference Machine produces an output fuzzy fact A' . And finally, it defuzzifies the output fuzzy fact A' to produce the real output a_0 .

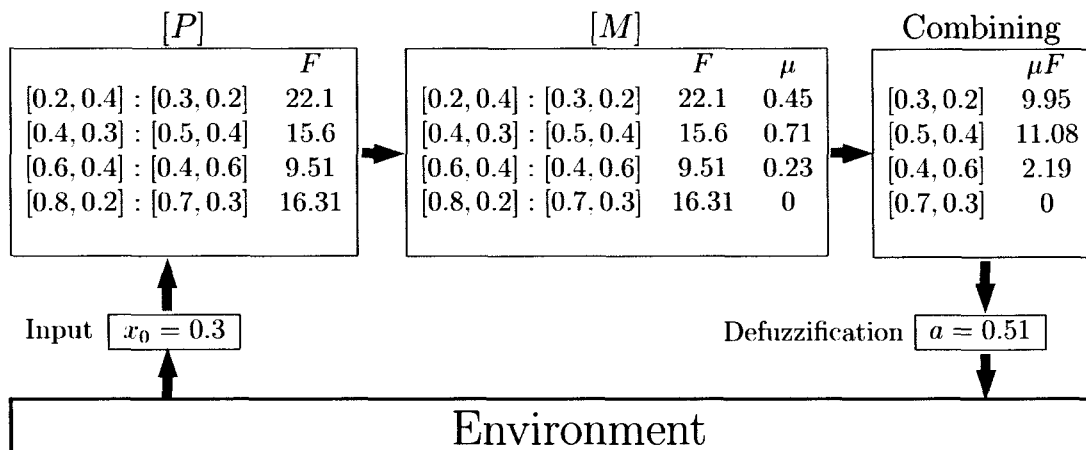


Figure 4.10: Diagram of the performance component of the Parodi and Bonelli's FCS.

- **Fuzzification.** It represents the input real vector $\vec{x}_0 = (x_1^0, x_2^0, \dots, x_n^0)$ into a set of values μ_i^l computed as

$$\mu_i^l = \mu_{X_i^l}(x_i^0). \quad (4.46)$$

Each fuzzy value X_i^l in rule R_i has its μ_i^l .

- **Inference Machine.** A match set $[M]$ is form from all of the rules $R_i \in [P]$ that have $\mu_i^l \neq 0$ for all $i \in \{1, 2, \dots, n\}$. The fuzzy fact A' of rule $R_i \in [M]$ is form as

$$\mu_{A'}(a) = \sum_{R_i \in [M]} \mu_i F_i \mu_{A'}(a), \quad (4.47)$$

where

$$\mu_i = \min [\mu_1^l, \mu_2^l, \dots, \mu_n^l]. \quad (4.48)$$

- **Defuzzification.** The output fuzzy fact A' is defuzzified by the center of gravity (Eq. 2.41) to produce the real output $a = a_0$.

Figure 4.10 shows an example with a few classifiers that follows the procedure described.

- **Learning Component.** In each step of the performance component, the environment gives to FCS a reward R . This reward is always a positive number. Then the F_i of the matched rules $R_i \in [M]$ are modified as:

$$F_i \leftarrow F_i - \beta F_i + R, \quad (4.49)$$

where $\beta \in [0, 1]$.

- **Discovery Component.** A GA is used to create new rules. Two rules are selected according to their F_i . They are crossed-over and mutated with probabilities χ and μ respectively. Then, one of the offspring is inserted in $[P]$ in placed of the worse

rule. The worst rule is determined by the rule that has the lower value on F_l . Crossover is done interchanging fuzzy sets between the rules. Mutation adds a real random number in $[-\mu_0, \mu_0]$ to the values c_i^l and w_i^l (or c^l and w^l).

The FCS is similar to Valenzuela's FCS. The main differences are that it does not use messages and that the fuzzy sets are not fixed anymore. Thus, the fuzzy sets are also evolved. The FCS was applied to continuous functions, too. This problem is again single-step, continuous in the input and continuous in the output.

4.3.3 Bonarini's FCS

In this FCS [3], rules not only cooperate as it is the normal in FL [32], rules can also compete among them. It uses fuzzy states that are a combination of fuzzy sub-states. These sub-states are something similar to the minimal messages from Valenzuela's FCS [30, 31]. The states are used to separate the matched rules into a set of subsets. In these subsets, the rules can compete while the best rules of each subset cooperate to produce the output fuzzy state. This FCS is compound by a set of linguistic variables, a population of rules, a performance component, a learning component and a discovery component.

- **Linguistic Variables.** It has a series of linguistic variables $(\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n, \mathcal{A})$ that match with the corresponding continuous variables $(x_1, x_2, \dots, x_n, a)$ that represent the input and the output variables. In each variable \mathcal{X}_i (or \mathcal{A}), a set of N_i (or M) linguistic values $\{X_{i1}, \dots, X_{iN_i}\}$ (or $\{A_1, \dots, A_M\}$) is defined. $\mu_{X_{ir}}(x_i)$ ($\mu_{A_s}(a)$) are defined as symmetric trapezoidal functions given by:

$$\mu_{X_{ir}}(x_i) = \begin{cases} \left[\frac{1}{\Delta_i} \right] x_i - \left[\frac{\delta_{ir}}{\Delta_i} \right], & \text{if } \delta_{ir} \leq x_i < (\delta_{ir} + \Delta_i); \\ 1, & \text{if } (\delta_{ir} + \Delta_i) \leq x_i < (\delta_{ir} + 2\Delta_i); \\ - \left[\frac{1}{\Delta_i} \right] x_i + \left[\frac{\delta_{ir} + 3\Delta_i}{\Delta_i} \right], & \text{if } (\delta_{ir} + 2\Delta_i) \leq x_i < (\delta_{ir} + 3\Delta_i); \\ 0, & \text{otherwise;} \end{cases} \quad (4.50)$$

$$\mu_{A_s}(a) = \begin{cases} \left[\frac{1}{\Delta} \right] a - \left[\frac{\delta_s}{\Delta} \right], & \text{if } \delta_s \leq a < (\delta_s + \Delta); \\ 1, & \text{if } (\delta_s + \Delta) \leq a < (\delta_s + 2\Delta); \\ - \left[\frac{1}{\Delta} \right] a + \left[\frac{\delta_s + 3\Delta}{\Delta} \right], & \text{if } (\delta_s + 2\Delta) \leq a < (\delta_s + 3\Delta); \\ 0, & \text{otherwise;} \end{cases} \quad (4.51)$$

where

$$\Delta_i = \frac{x_i^{\max} - x_i^{\min}}{2N_i - 1}; \quad \Delta = \frac{a^{\max} - a^{\min}}{2M - 1}, \quad (4.52)$$

$$\delta_{ir} = x_i^{\min} + (2r - 3)\Delta_i; \quad \delta_s = a^{\min} + (2s - 3)\Delta. \quad (4.53)$$

with $[x_i^{\min}, x_i^{\max}]$ (or $[a^{\min}, a^{\max}]$) the lower and the upper limits of the variable x_i (or a). Figure 4.11 shows the membership functions of these linguistic variables.

- **Population of Rules.** This FCS has a fixed population of rules $[P]$. Each rule R_l has a condition, an action, an activity level μ_l , and strength F_l as it follows:

$$\text{IF } [\mathcal{X}_1 = X_i^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A} = A^l]; \quad \mu_l; \quad F_l, \quad (4.54)$$

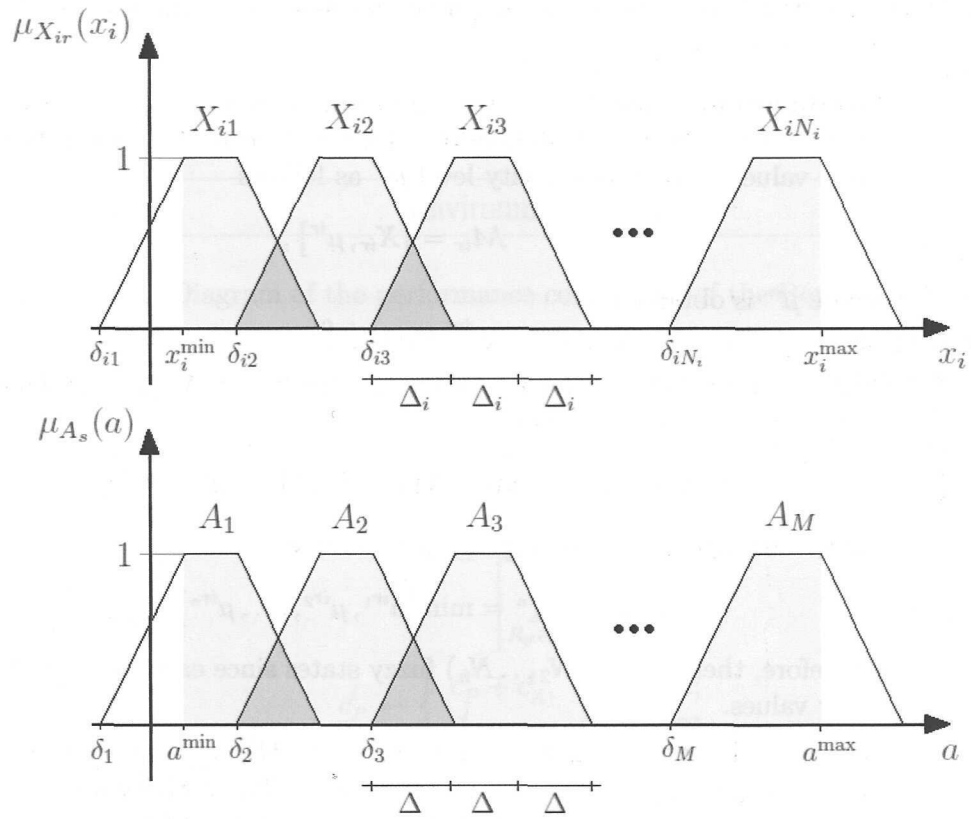


Figure 4.11: Membership functions of a linguistic variable in Bonarini's FCS

where X_i^l and A^l denotes a fuzzy value of \mathcal{X}_i in the l -th rule that can also be the symbol $\#$ and a fuzzy value of \mathcal{A} in the l -th rule. The symbol $\#$ means that any fuzzy value can match the corresponding variable. These rules are coded as:

$$2, 4, \#, 5, 2, \dots, \#, \# : 2; \mu_i; F_i, \quad (4.55)$$

where the commas represents AND connectors, the numbers the fuzzy values of the corresponding linguistic variables, and the colon the separation between the condition and the action.

- **Performance Component.** This component determines the way the FCS acts. It repeats the following steps:

- **Fuzzification.** First, the input real vector $\vec{x}_0 = (x_1^0, x_2^0, \dots, x_n^0)$ is used to create a set of fuzzy sub-states \mathcal{M}_{ir} . Each sub-state \mathcal{M}_{ir} is compound as a fuzzy value X_{ir} and an activity level μ^{ir} as follows

$$\mathcal{M}_{ir} = [X_{ir}, \mu^{ir}], \quad (4.56)$$

where μ^{ir} is obtained as

$$\mu^{ir} = \mu_{X_{ir}}(x_i^0). \quad (4.57)$$

The sub-states are combined to generate fuzzy states $\mathcal{N}_{r_1, r_2, \dots, r_n}$. These fuzzy states have the next structure:

$$\mathcal{N}_{r_1, r_2, \dots, r_n} = [\mathcal{M}_{1r_1}, \mathcal{M}_{2r_2}, \dots, \mathcal{M}_{nr_n}, \mu^{r_1, r_2, \dots, r_n}], \quad (4.58)$$

where each $r_i \in \{1, 2, \dots, N_n\}$ and $\mu^{r_1, r_2, \dots, r_n}$ is

$$\mu^{r_1, r_2, \dots, r_n} = \min [\mu^{ir_1}, \mu^{ir_2}, \dots, \mu^{ir_n}]. \quad (4.59)$$

Therefore, there are $(N_1 N_2 \dots N_n)$ fuzzy states since each variable \mathcal{X}_i has N_i fuzzy values.

- **Inference Machine.** A set of matched rules $[M]_{r_1, r_2, \dots, r_n}$ is created for each fuzzy state $\mathcal{N}_{r_1, r_2, \dots, r_n}$ that has $\mu^{r_1, r_2, \dots, r_n} \neq 0$. The activity level μ_i of rules $R_i \in [M]_{r_1, r_2, \dots, r_n}$ are set equal to the activity level of the fuzzy state $\mu^{r_1, r_2, \dots, r_n}$. So the matching set $[M]$ is form by the union of all of the $[M]_{r_1, r_2, \dots, r_n}$. A rule R_{r_1, r_2, \dots, r_n} from each $[M]_{r_1, r_2, \dots, r_n}$ is selected. R_{r_1, r_2, \dots, r_n} is the one with the highest F_i . These selected rules R_{r_1, r_2, \dots, r_n} form a set of rules $[F]$. An action set $[A]$ is formed with all of the rules from sets $[M]_{r_1, r_2, \dots, r_n}$ that propose the same actions as rules $R_i \in [F]$. The fuzzy fact A' is obtained by:

$$\mu_{A'}(a) = \sum_{R_i \in [A]} \eta \mu_i \mu_{A'}(a), \quad (4.60)$$

where $\eta \in [0, 1]$.

- **Defuzzification.** The output fuzzy fact A' is defuzzified by the center of gravity (Eq. 2.41) to produce the real output $a = a_0$.

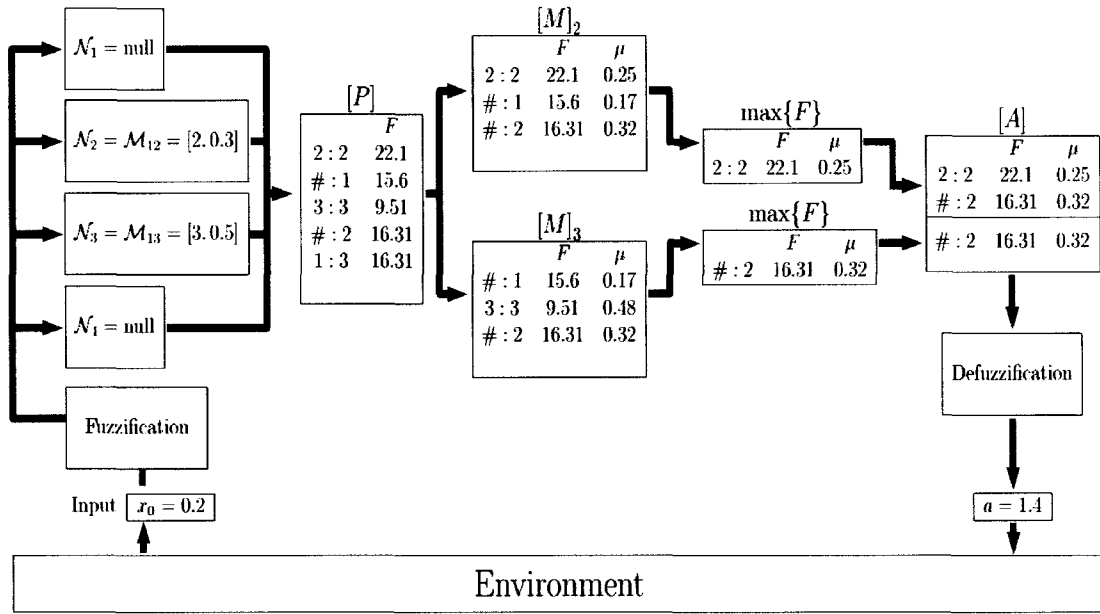


Figure 4.12: Diagram of the performance component of the Bonarini's FCS.

Figure 4.12 shows an example with a few classifiers that follows the procedure described.

- **Learning Component.** The actual and the past contributions of the rule $R_i \in [F]$ are defined as:

$$c_A^j = \frac{\mu_i}{\left[\sum_{R_{i'} \in [F]} \mu_{i'} \right]}, \quad (4.61)$$

$$c_P^j \leftarrow \begin{cases} c_P^j + c_A^j, & \text{if } c_P^j \leq \varepsilon; \\ c_P^j, & \text{otherwise.} \end{cases} \quad (4.62)$$

Now, there are four possible ways of applying learning that have been studied:

1. The F_i of all rules $R_i \in [F]_t$ are modified using the reward R given by the environment as

$$F_i \leftarrow F_i + \left[\frac{c_A^j}{c_P^j} \right] (R - F_i). \quad (4.63)$$

2. This is a version of the Bucket-Brigade Algorithm. First, all the F_i in the rules $R_i \in [A]_t$ are modified as follows:

$$F_i \leftarrow F_i - \beta c_A^j F_i, \quad (4.64)$$

where $\beta \in [0, 1]$. At the same time, the F_i of all of the rules $R_i \in [A]_{t-1}$ are modified as follows:

$$F_i \leftarrow F_i + \left[\frac{c_A^j}{c_P^j} \right] (B_i + R), \quad (4.65)$$

with

$$B_t = \sum_{R_i \in [A]_t} \beta c_A^i F_i, \quad (4.66)$$

where R is the reward at time t .

3. It uses Q-learning. The F_i of the rules $R_i \in [A]_{t-1}$ are modified as follows:

$$F_i \leftarrow F_i + \beta \left[\frac{c_A^i}{c_P^i} \right] \left[\left(R_{t-1} + \gamma \max_{R_{i'} \in [A]_t} F_{i'} \right) - F_i \right], \quad (4.67)$$

where $\beta \in [0, 1]$, $\gamma \in [0, 1]$ and R_{t-1} is the reward obtained at time $(t - 1)$.

4. It uses Q-learning but in a different way. The F_i of the rules $R_i \in [A]_{t-1}$ is modified as follows:

$$F_i \leftarrow F_i + \beta c_A^i \left[\left(R_{t-1} + \gamma \sum_{R_{i'} \in [F]_t} \mu_{i'} F_{i'} \right) - F_i \right], \quad (4.68)$$

where $\beta \in [0, 1]$, $\gamma \in [0, 1]$.

- **Discovery Component.** A GA is used to create new rules on $[P]$ (or $[M]$). Two rules are selected according to their F_i , they are crossed-over and mutated with probabilities χ and μ , respectively, and finally one of the offspring is inserted on $[P]$ (or $[M]$) in place of the worse rule. The worse rule is the rule that has the lowest value of F_i .

Bonarini's FCS introduces the idea of making competition and cooperation among rules in different stages of the algorithm. The key ingredient is that classifiers that represent fuzzy rules do not form a fuzzy system but a subset of them. This subset is always different and begins by the fuzzification of the state of the problem into a set of internal fuzzy states. For each fuzzy state an activation set is created. Rules in each activation set compete to select one. Those selected rules, one per each activation set, form the fuzzy set and, thus, cooperate to create the output.

This FCS was applied in a navigation task with a CAT robot moving in a corridor. The robot is moving with constant speed through the corridor. The task is to maintain the robot in the center of the corridor. The problem is shown in Fig. 4.13. The reward function is continuous and depends on the distance between the robot and the center. The goal is the line in the center of the corridor. This problem is continuous in its states and in its actions. A more complex problem would be the one with reward only in the goal. This means that with continuous reward the robot is guided in the learning and with reward only over the goal, the robot has to discover how to keep itself in the center.

4.4 General Discussion

LCSs described in this chapter were selected by the elements they introduced that are fundamental for this research. They were also taken as the foundations of the

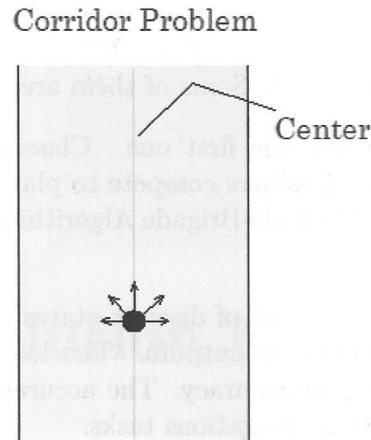


Figure 4.13: Corridor problem with a CAT robot.

LCSs. Holland's LCS was the first one to introduce the idea. Some researchers make research about LCSs because their capabilities of adaptation and learning. This has taken researchers to consider this could be how our brains work. In a simplified view, our brains learn about the world adjusting the weights of the neural nets, but also they change those connections creating new ones and eliminating others. Nobody knows how this is done precisely but it seems to be an adaptable system that is able to learn. Due to this parallelism with LCSs, LCSs have called attention. But the classifiers of LCSs also act in parallel and can make generalizations. These characteristics make LCSs good system for many applications [5] of the real world in such domains as data mining [6], modelling and optimization, and control.

As it was shown in this chapter, LCSs work well in discrete problems like Holland's LCS and XCS. But many other problems of the real life are not discrete. Due to this, LCSs have been changed to cover these limitations. A proof of this is XCSF that is able to deal with continuous inputs. The introduction of fuzzy logic allows LCSs to work with continuous inputs and with continuous outputs as well. The FCSs of Valenzuela and Parodi et al. only learn continuous functions while the one from Bonarini is capable of learning more complex problems defined with continuous inputs and outputs. These problems are the ones that require more than one action to solve the task and with a continuous reward function.

QFCS is different from these FCSs because QFCS classifiers are not fuzzy rules but fuzzy systems. This allows the introduction of a hyper-matrix to learn not only one parameter to approximate the Q-function but many. The problem with traditional FCSs is that they use a single parameter to approximate the Q-function. But fuzzy rules work in combination with other fuzzy rules that are not always the same ones in their neighborhoods. So, that value is not representative of the Q-function in those neighborhoods. Therefore, the introduction of the hyper-matrix in each fuzzy system is to learn directly the Q-function over the input state space. That is why QFCS associates the hyper-matrices to small areas in the activation regions of the classifiers. Therefore, it is very important to remark that QFCS is trying to learn the Q-function.

4.5 Summary

There are many approaches to LCS. Some of them are:

- HOLLAND's LCS. It was the first one. Classifiers associate a set of discrete states with an action. Classifiers compete to place their actions as outputs. The learning is done by the Buckade-Brigade Algorithm. A GA evolves classifiers based on strength.
- XCS. Classifiers associate a set of discrete states with an action. Classifiers compete to place their actions as outputs. The learning is by Q-learning. A GA evolves classifiers based on accuracy. The accuracy measures the convergence of the Q-values. XCS solved navigations tasks.
- XCSF. Classifiers associate a set of continuous states with an action. Classifiers compete to place their actions as outputs. The learning is by Q-learning. Classifiers approximate the Q-function through dot products formed by the input vector with a weight vector. A GA evolves classifiers based on accuracy. The accuracy measures the convergence of the Q-values. XCSF learned continuous functions and solved navigations tasks.
- VALENZUELA's FCS. Each classifier is a fuzzy rule. The FCS is a fuzzy system. Therefore, the FCS associates a set of continuous states with another set of continuous actions. The fuzzy sets are fixed. Rules cooperate to create the output. Learning is based on the Buckade-Brigade Algorithm. A GA evolves classifiers based on their strength. FCS learned continuous functions.
- PARODI AND BONELLI's FCS. It is similar to Valenzuela's FCS but with unfixed fuzzy sets that are also evolved by the GA.
- BONARINI's FCS. Each classifier is a fuzzy rule. The fuzzy sets are also fixed. Rules cooperate and compete to create the output. Learning is based on Q-learning and by the Buckade-Brigade Algorithm. A GA evolves classifiers based on their strength. FCS was applied in the movement of a CAT robot in corridor with a continuous reward function.

Chapter 5

Proposed Solution Model

This chapter describes QFCS [21, 22, 23]. QFCS is a Fuzzy Classifier System that is able to learn by reward in continuous spaces with a set of continuous vector actions. Fuzzy Logic [32] is used as the main representation of classifiers. Each classifier is a Small Fuzzy System (SFS). This SFS represents a relationship between input and output variables. That relationship is a vector field. Q-Learning [27, 24] is used to provide QFCS with the ability of learning by reward. Classifiers approximate the Q-function by hyper-matrices over the vector fields. Therefore, QFCS is learning the Q-function. Two different QFCS are described, one with fixed fuzzy sets and another with unfixed fuzzy sets. QFCS with unfixed fuzzy sets was introduced as a generalization with the hope of having classifiers better adjusted to the solution. These QFCSs were tested in simple continuous problems with different levels of difficulty. These problems are the Frog Problem, the n -Environment Problem in one and two dimensions that represent navigation tasks, and two different versions of the Inertial Particle Problem in one dimension. All of them will be described in the next chapter.

5.1 QFCS with Fixed Fuzzy Sets

In QFCS [21, 22], each classifier contains a small fuzzy system (SFS), a matrix containing the expected prediction, and a square sub-region of activation over the input space. The classifiers can only act over their sub-region of activation. In that sub-region of activation, each fuzzy system proposes a continuous vector field as an action by defuzzification. In that way, when an input vector enters the QFCS, classifiers compete to place their actions according to their expected predictions. A Q-learning algorithm is used to learn the task from the environment, i.e., the learning of the continuous vector action function. This algorithm is used to change the values of the matrices of the expected predictions for each classifier in the QFCS. A GA is applied to evolve rules based on their average expected predictions. This GA evolves only the action parts of the fuzzy systems.

The components of the QFCS, that are shown in Fig. 5.1, are a set of N classifiers defined over n input variables and m output variables, a performance component, a learning component, and a discovery component:

Chapter 5

Proposed Solution Model

This chapter describes QFCS [21, 22, 23]. QFCS is a Fuzzy Classifier System that is able to learn by reward in continuous spaces with a set of continuous vector actions. Fuzzy Logic [32] is used as the main representation of classifiers. Each classifier is a Small Fuzzy System (SFS). This SFS represents a relationship between input and output variables. That relationship is a vector field. Q-Learning [27, 24] is used to provide QFCS with the ability of learning by reward. Classifiers approximate the Q-function by hyper-matrices over the vector fields. Therefore, QFCS is learning the Q-function. Two different QFCS are described, one with fixed fuzzy sets and another with unfixed fuzzy sets. QFCS with unfixed fuzzy sets was introduced as a generalization with the hope of having classifiers better adjusted to the solution. These QFCSs were tested in simple continuous problems with different levels of difficulty. These problems are the Frog Problem, the n -Environment Problem in one and two dimensions that represent navigation tasks, and two different versions of the Inertial Particle Problem in one dimension. All of them will be described in the next chapter.

5.1 QFCS with Fixed Fuzzy Sets

In QFCS [21, 22], each classifier contains a small fuzzy system (SFS), a matrix containing the expected prediction, and a square sub-region of activation over the input space. The classifiers can only act over their sub-region of activation. In that sub-region of activation, each fuzzy system proposes a continuous vector field as an action by defuzzification. In that way, when an input vector enters the QFCS, classifiers compete to place their actions according to their expected predictions. A Q-learning algorithm is used to learn the task from the environment, i.e., the learning of the continuous vector action function. This algorithm is used to change the values of the matrices of the expected predictions for each classifier in the QFCS. A GA is applied to evolve rules based on their average expected predictions. This GA evolves only the action parts of the fuzzy systems.

The components of the QFCS, that are shown in Fig. 5.1, are a set of N classifiers defined over n input variables and m output variables, a performance component, a learning component, and a discovery component:

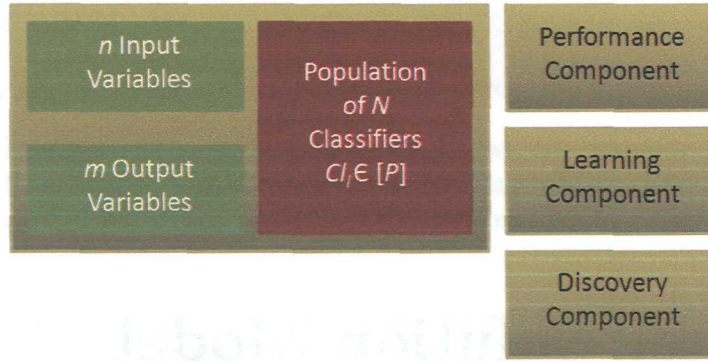


Figure 5.1: Diagram of components of the QFCS

- **Classifiers.** Each classifier Cl_l contains a SFS_l over the input $x_1 \dots x_n$ and output $a_1 \dots a_m$ spaces, and a square region R^l over that input space defined as

$$R^l = \{ (x_1, \dots, x_n) \mid (\min_{x_1}^l \leq x_1 \leq \max_{x_1}^l) \wedge \dots \wedge (\min_{x_n}^l \leq x_n \leq \max_{x_n}^l) \}, \quad (5.1)$$

where $\min_{x_i}^l$ and $\max_{x_i}^l$ with $i = 1, \dots, n$ are constants, and a set of elements p_{d_1, \dots, d_n}^l with the subscripts $d_i \in \{1, \dots, d\}$, $i = 1, \dots, n$ and d a constant that constitutes an n -dimensional matrix. The classifiers can act only over their regions R^l . To form these regions, the whole input space is divided into c^n (c to the power of n) uniform square regions R_{c_1, \dots, c_n} with the subscripts $c_i \in \{1, \dots, c\}$, $i = 1, \dots, n$ and c a constant that determines the number of divisions per dimension. Then, classifiers can only take one out of them. Each R_{c_1, \dots, c_n} is again divided into d^n (d to the power of n) uniform square regions Δ_{d_1, \dots, d_n} that are associated one by one with p_{d_1, \dots, d_n}^l . Figure 5.2 shows an example with $n = 2$, $c = 4$ and $d = 4$. In figure 5.2.a it can be seen, in light gray, the input space that is a two dimensional space. Figure 5.2.b represents the input space divided in $c = 4$ square regions. One of these regions, region R_{43} in medium gray, is taken by the classifier Cl_l . Figure 5.2.c shows how the region R_{43} is divided again into $d = 4$ square uniform sub-region. This division is of the same size of the Cl_l 's matrix. Each element of the matrix is then associated with each respective sub-region one by one. In dark gray, as an example, it is represented the element Δ_{32} that is associated with the element p_{32}^l . These p_{d_1, \dots, d_n}^l elements represent the Q function $Q(x_1, \dots, x_n, a_1, \dots, a_m)$. Since each classifier has a SFS_l , each Cl_l represents a continuous vector field over the region R^l . This vector function is formed by the SFS_l of each Cl_l . Therefore, each component of that vector field is a continuous function $a_j^l(x_1 \dots x_n)$. Figure 5.2.d shows the j -th component of the vector field for Cl_l .

- **Performance Component.** This component determines the procedure QFCS follows to select actions as responses to the inputs. First, it receives a real vector \vec{x}_0 as input; then, all classifiers that contain the input in their own regions are activated to form a match set $[M]$. At that time, the input is fuzzified and is introduced into each SFS_l of the classifiers in $[M]$. The fuzzy inference machine of

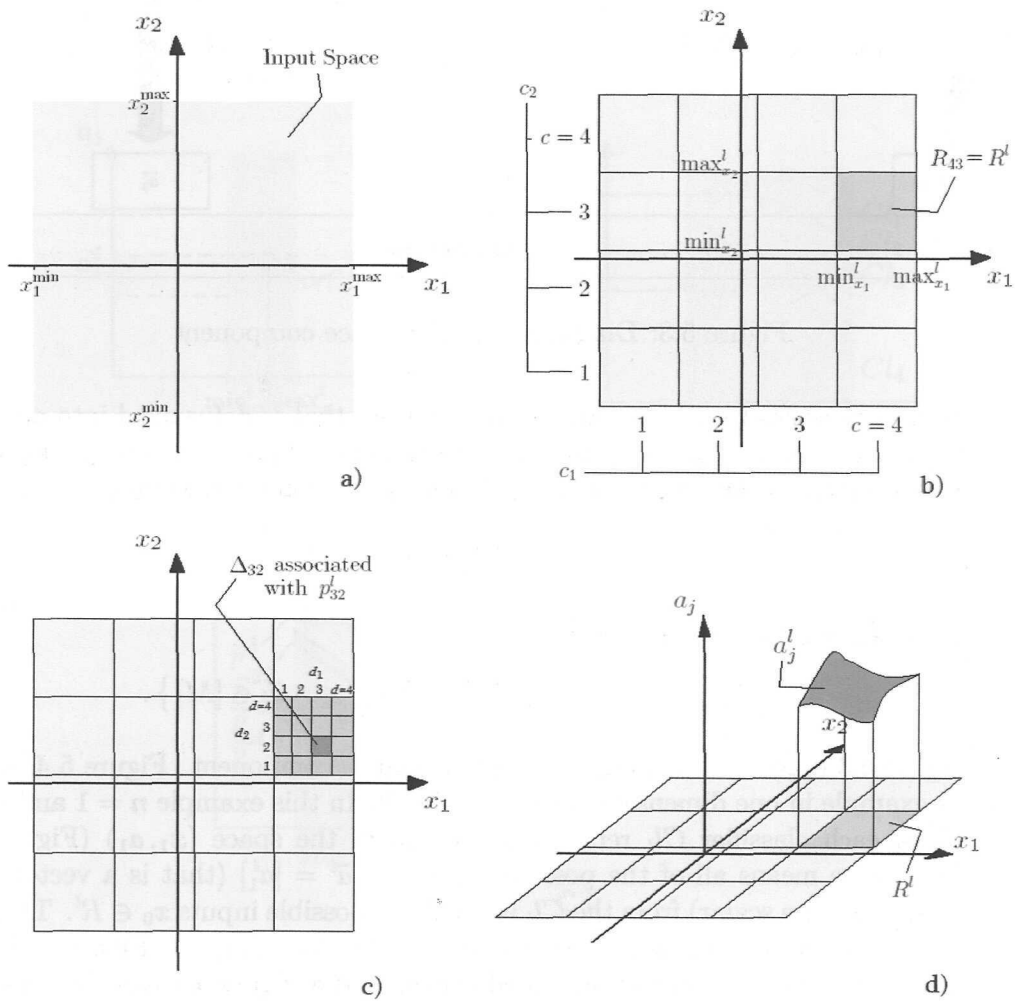


Figure 5.2: a. Input space $x_1 \dots x_n$ with $n = 2$. b. Square regions R_{c_1, \dots, c_n} with $c = 4$. c. Square sub-regions Δ_{d_1, \dots, d_n} with $d = 4$. d. j -th component of the vector field $\vec{a}_l = (a_1^l, \dots, a_m^l)$

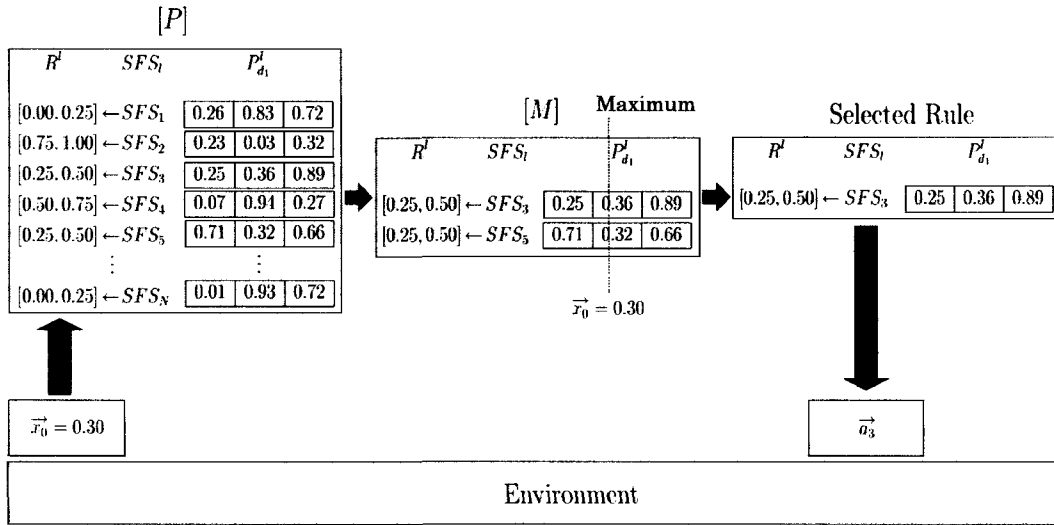


Figure 5.3: Diagram of performance component

each SFS_l works to produce an output fuzzy set that is defuzzified into an output vector $\vec{a}_i = (a_1^i, \dots, a_m^i)$ proposed by each classifier Cl_i in $[M]$. Then, one of those \vec{a}_i is selected as the output of the QFCS \vec{a}_0 in the following manner:

$$\vec{a}_0 = \arg \max_{\vec{a}_i \in [M]} [P_{d_1, \dots, d_n}], \quad (5.2)$$

where P_{d_1, \dots, d_n} is a set defined by

$$P_{d_1, \dots, d_n} = \{p_{d_1, \dots, d_n}^l \mid \vec{x}_0 \in \Delta_{d_1, \dots, d_n} \wedge \vec{a}_i \in [M]\}. \quad (5.3)$$

Figure 5.3 depicts a diagram of the performance component. Figure 5.4 presents an example in one dimension with more detail. In this example $n = 1$ and $m = 1$. Thus, each classifier Cl_i represents a curve in the space (x_1, a_1) (Fig. 5.4.a). This curve means all of the possible responses $\vec{a}^i = [a_1^i]$ (that is a vector of one component, an scalar) from the Cl_i to all of the possible inputs $x_0 \in R^l$. Therefore, the system can be seen as a series of curves in the space (x_1, a_1) . Figure 5.4b shows the performance component described above. In that figure one can see 4 classifiers represented as red curves with their corresponding prediction vectors. The input x_0 cuts all the classifiers Cl_1, Cl_2, Cl_3 and Cl_4 forming the red points. This input also cuts the prediction hyper-matrices $\vec{p}^1, \vec{p}^2, \vec{p}^3$ and \vec{p}^4 , that are vectors now, in the second components p_2^1, p_2^2, p_2^3 , and p_2^4 . These classifiers Cl_1, Cl_2, Cl_3 and Cl_4 propose the outputs a_1^1, a_1^2, a_1^3 and a_1^4 that compete depending of which prediction value p_2^1, p_2^2, p_2^3 , and p_2^4 is maximal to form the output of the system a_0 . The prediction vectors can be seen in Fig. 5.4c. They can be placed over the rules. It means that each value represents the expected prediction of the system over a little piece of the curve of a rule. Following this way of thinking, it is represented the Q-values of the Q-function from Q-learning only over little pieces of lines, not in the complete domain.

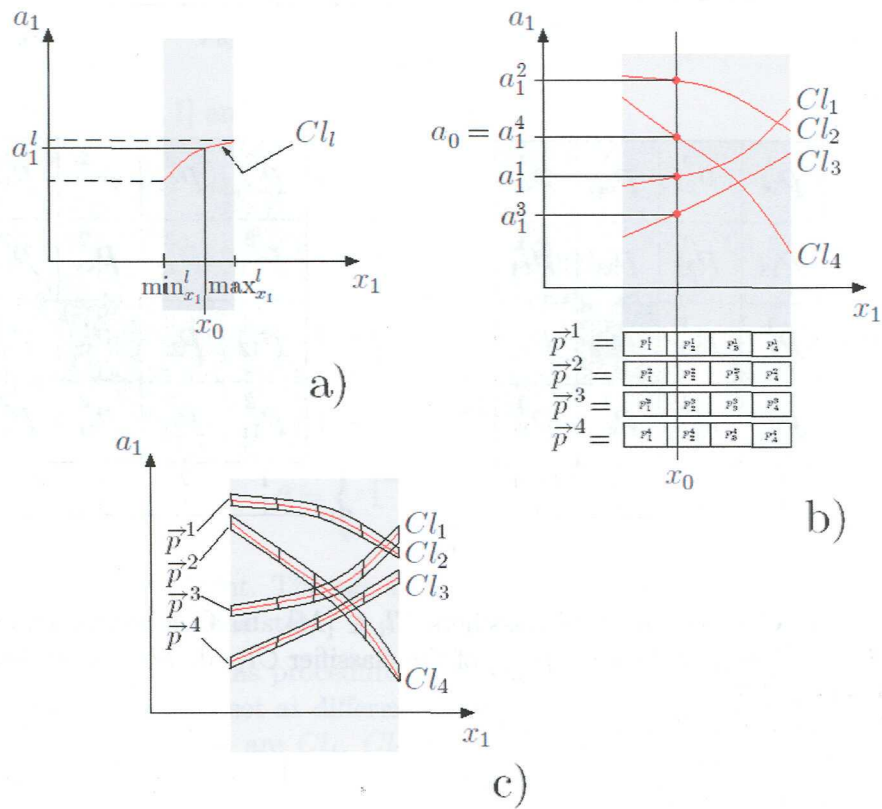


Figure 5.4: a. Meaning of a classifier. b. Performance component. c. Meaning of the prediction vectors

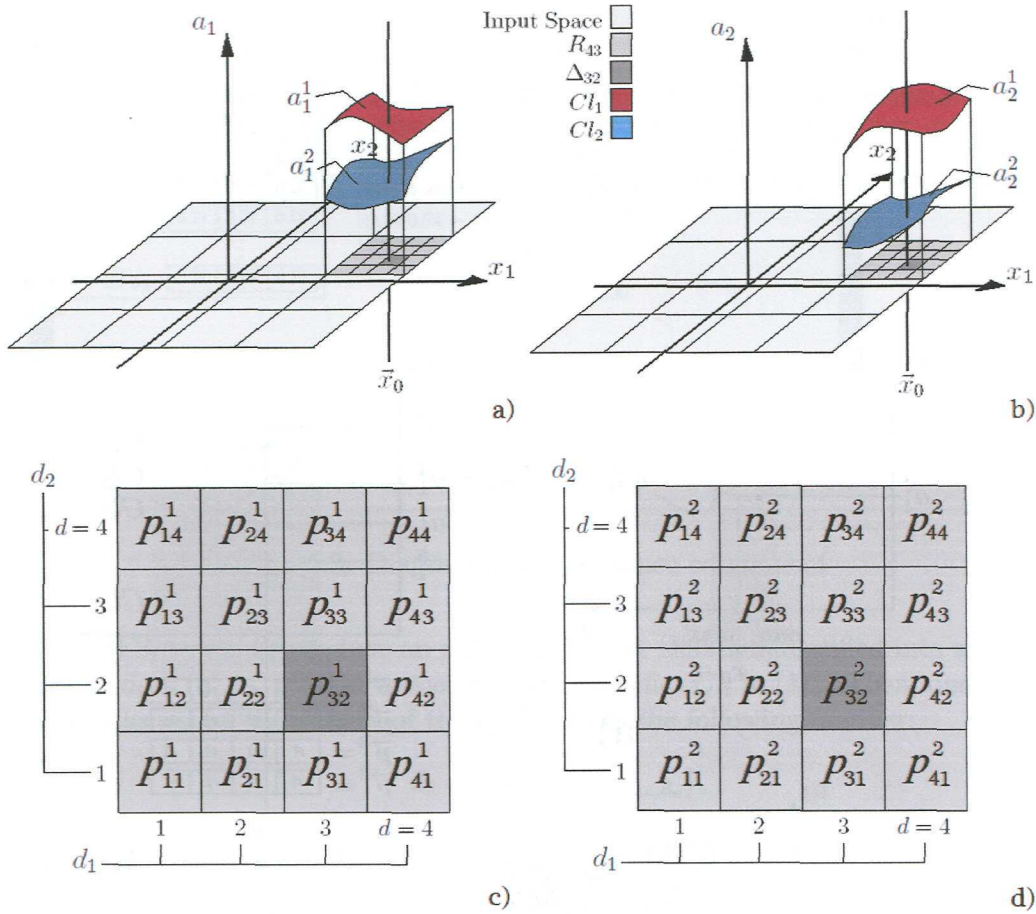


Figure 5.5: a. Component a_1^l of classifiers $Cl_l \in [M]$. b. Component a_2^l of classifiers $Cl_l \in [M]$. c. Matrix of elements p_{d_1,d_2}^1 of the classifier Cl_1 . d. Matrix of elements p_{d_1,d_2}^2 of the classifier Cl_2 .

Figure 5.5 presents another example in two dimensions with detail. In this example $n = 2$ and $m = 2$. Figures 5.5.a and 5.5.b show the components of the output vectors of the SFS $_l$ s in $[M]$. There are only 2 classifiers in $[M]$ in this example and they are called for simplicity classifiers 1 and 2. Surfaces with the same color belong to the same classifier. In that picture, the input vector \vec{x}_0 is observed. That vector pass through sub-region Δ_{32} of region R_{43} and pass through the surfaces of the classifiers determining their output vectors \vec{a}_1 and \vec{a}_2 . Figures 5.5.c and 5.5.d represent the matrices of prediction values p_{d_1,d_2}^1 and p_{d_1,d_2}^2 of Cl_1 and Cl_2 , respectively. These matrices can be seen as placed over the surfaces. Both of the component of the same classifier share the same matrix. It is also seen that the elements p_{32}^1 and p_{32}^2 are the ones that satisfy 5.2. This means these values are associated with Δ_{32} and Δ_{32} contains the input vector \vec{x}_0 . So, if $p_{32}^1 \geq p_{32}^2$ then Cl_1 places its output vector as the output vector of the QFCS \vec{a}_0 and vice versa.

- **Learning Component.** This component makes the QFCS learn the task using Q-learning. QFCS receives, from the environment, a reward $R(\vec{x})$ that defines the task to achieve. At each time, QFCS decides what to do, exploiting the knowledge it has acquired or exploring new possible actions. The exploitation is done with probability P_E , while exploration is done with probability $P_R = 1 - P_E$. Exploitation is achieved by selecting \vec{a}_0 as in the performance component (Eq. 5.2), and exploration by selecting one of the possible $\vec{a}_i \in [M]$ at random. Each time t , the expected prediction values p_{d_1, \dots, d_n}^l at time $(t - 1)$ are adjusted as follows:

$$p_{d_1, \dots, d_n, t-1}^l \leftarrow p_{d_1, \dots, d_n, t-1}^l + \beta \left[\left(R(\vec{x}_{t-1}) + \gamma p_{d_1, \dots, d_n, t}^l \right) - p_{d_1, \dots, d_n, t-1}^l \right], \quad (5.4)$$

where $\beta, \gamma \in [0, 1]$ and

$$p_{d_1, \dots, d_n, t-1}^l = \{ p_{d_1, \dots, d_n}^l | \vec{x}_{t-1} \in \Delta_{d_1, \dots, d_n} \wedge Cl_l = Cl_{t-1} \}, \quad (5.5)$$

$$p_{d_1, \dots, d_n, t}^l = \{ p_{d_1, \dots, d_n}^l | \vec{x}_t \in \Delta_{d_1, \dots, d_n} \wedge Cl_l = Cl_t \}, \quad (5.6)$$

with \vec{x}_{t-1} and Cl_{t-1} that are the input and the classifier that was selected by the performance component at time $(t - 1)$, and \vec{x}_t and Cl_t are the input and the classifier that would be selected by exploitation at time t . β is variable throughout time depending on how old δ_l , the classifier Cl_l , is. In other words:

$$\beta = \begin{cases} \left[\frac{\beta_0 - 1}{\delta_0} \right] \delta_l + 1, & \text{if } \delta_l < \delta_0; \\ \beta_0, & \text{otherwise,} \end{cases} \quad (5.7)$$

where δ_0 is a constant. The age δ_l of the classifiers starts in 0 and is incremented in one unit every time step.

Figure 5.6 depicts this procedure in an example with $n = 1$ and $m = 1$. There is shown the match set at different times. In the left are the classifiers activated at time $(t - 1)$ that are Cl_1, Cl_2, Cl_4 and Cl_4 . In the right are the classifiers activated at time t that are Cl_5, Cl_6, Cl_7 and Cl_8 . The input values x_{t-1} and x_t cut with lines the classifiers and the prediction vectors. Red points represent the action proposed by the classifiers. The action of the Cl_3 , that is $a_0 = a_1^3$ is the action selected by QFCS at time $(t - 1)$. The action of the Cl_6 , that is a_1^6 is the one that has the maximal prediction value at time t . This is

$$a_1^6 = \arg \max_{a_1} [p_3^5, p_3^6, p_3^7, p_3^8] \quad (5.8)$$

Then, p_3^6 is used to modify p_2^3 that belongs to Cl_3 .

Figure 5.7 shows another example with $n = 2$ and $m = 1$. Figure 5.7.a represents the match set $[M]$ at time $(t - 1)$. It contains only 3 classifiers called Cl_1, Cl_2 and Cl_3 . Figure 5.7.b represents the match set $[M]$ at time t . It contains also 3 classifiers called Cl_4, Cl_5 and Cl_6 . The matrices of the prediction values are represented over the surfaces of each classifier as a grid. In those grids, there are

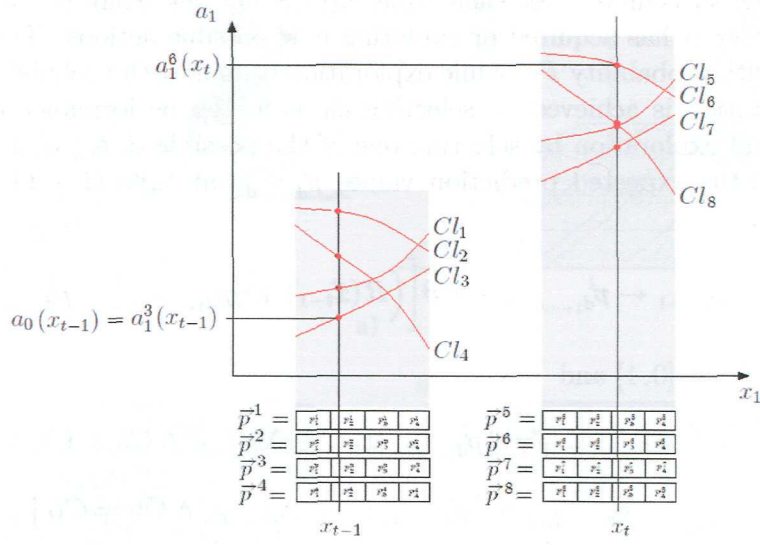


Figure 5.6: Learning component

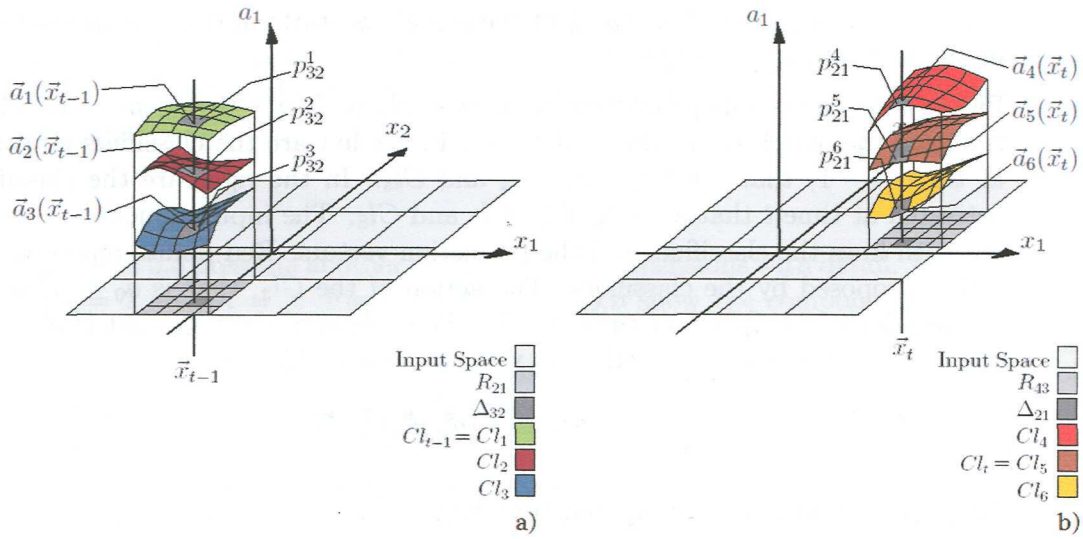


Figure 5.7: a. QFCS at time $(t - 1)$. b. QFCS at time t .

regions drawn in dark gray that have the prediction values of interest. Classifier Cl_1 is the one selected at time $(t - 1)$ while classifier Cl_5 is the one that would be selected at time t by exploitation. The difference between both of these classifiers is that, the former can be selected either by exploitation or by exploration while the latter is always selected by exploitation. With these conditions, the element p_{21}^5 of the classifier Cl_5 is used to update the element p_{32}^1 of the classifier Cl_1 according to 5.4. Note that p_{21}^5 is associated with Δ_{21} in R_{43} and that $\vec{x}_t \in \Delta_{21}$. In the same way, p_{32}^1 is associated with Δ_{32} in R_{21} and that $\vec{x}_{t-1} \in \Delta_{32}$. As it can be seen, this procedure is similar to Q-learning. The difference is that in this case QFCS is trying to represent with its rules a discrete Q function over certain regions (the surfaces of the classifiers) of the continuous space $x_1 \dots x_n a_1 \dots a_m$ and not over a discrete space.

- **Discovery Component.** QFCS uses a GA to create new rules. The GA is applied over $[M]$ at time before $(t - 1)$. It only evolves the action parts of the fuzzy rules of the SFS $_i$ s. First, the \bar{p}_i value of each $Cl_i \in [M]$ is computed by averaging their expected values p_{d_1, \dots, d_n}^i . Then the GA takes two classifiers from $[M]$ selecting them with probability proportional to their \bar{p}_i . These classifiers are copied, crossed-over with probability χ , and mutated with probability μ . One-point crossover is used. Mutation changes a 0 by a 1 or vice versa. Then, one of the two classifiers is inserted in $[M]$ replacing one in $[M]$ that is selected proportional to b/\bar{p}_i . The GA is applied over $[M]$ on time intervals that are determined by the classifiers in $[M]$. Each classifier stores the last time e_i in which it was involved in a GA, so when $[M]$ happens, the average time \bar{e} of its classifiers, for the last application of the GA, is calculated by:

$$\bar{e} = \frac{\sum_{Cl_i \in [M]} (t - e_i)}{|[M]|}, \quad (5.9)$$

where $|[M]|$ represents the number of classifiers in $[M]$. If $\bar{e} \geq \theta_{GA}$ the GA is applied. This part evolves the surfaces of the classifiers to those that represent the higher Q-values. These surfaces end up covering the most important regions of the space for the task. A classifier by itself can not represent all the solution in its activation region. Therefore all the classifiers in that activation region define a piecewise solution. This is the approach of the Learning Classifier Systems.

Each classifier in QFCS has a SFS. This SFS is defined over the activation region of the classifier. The SFS defines a relationship between the input and the output variables. In general, QFCS has n input variables and m output variables. So, the SFS represents a continuous vector field. The components of the vector field are each one a scalar function over the input variables. These scalar functions are functions of n variables. Thus, if $n > 3$, the functions are known as hyper-functions. This is very important because those hyper-functions define some points in the joined input-output space that are related by the fuzzy system. It is over those points where QFCS learns the Q-function through a hyper-matrix in n dimensions associated to those points defined

by the hyper-functions. The fact that the activation regions are fixed and that the SFSS do not change in its structure follows that the input fuzzy sets are also fixed.

In QFCS, classifiers combine the fuzzy systems with Q-learning through the hyper-matrices. Therefore, each classifier represents hyper-functions that define hyper-surfaces. The hyper-matrices represent the Q-values over those hyper-surfaces. In this way, each classifier tries to learn the Q-function over those hyper-surfaces. Those hyper-surfaces do not represent all of the input-output space but just a small sub-region. Since, there are many classifiers, it is possible to cover all input-output space. It is important to note that QFCS evolves classifiers moving the hyper-surfaces to those places where the Q-function is higher and because of that is that QFCS finishes with a mapping over the regions in the input-output spaces that are important for making decisions.

The parameters to be defined are in Table 5.1. Since the activation regions of classifiers are fixed. The number of classifiers that contain a determined activation region do not change over time because the GA only evolves the action parts of the fuzzy rules. This number is about $N_{AR} \approx N/c^n$ where c^n is the number of activation regions. Thus, N_{AR} has to be of enough size to allow the GA to evolve classifiers. The bigger N_{AR} , the better. QFCS uses $N_{AR} \approx 50$. Therefore, the number of the population of QFCS is $N \approx N_{AR}c^n = 50c^n$. Parameter d defines the number of elements per dimension of the hyper-matrices. This value has to be small enough to allow a well learning of the Q-function. If $d = 1$ then each classifier has one Q-value to approximate the Q-function which is too optimistic. If $d \rightarrow \infty$ is bigger, QFCS would take too much time in learning the Q-function with that precision. Therefore, QFCS uses $d \approx 5$. Parameter c has to be small enough to allow hyper-surfaces connect following the curvature of the solution. Curvature is constructed by the connection of many hyper-surfaces. If $c = 1$ then the connection among hyper-surfaces has almost no curvature which is also too optimistic. If $c \rightarrow \infty$ allows hyper-surfaces connect with high curvatures. Therefore, QFCS uses $c \approx 4$. This parameter determines which problems are easy for QFCS depending on the curvature of the solution, but its elimination let the activation regions be free or be unfixed allowing hyper-surfaces to follow the curvature of the solution. Because of this reason, QFCS with the unfixed fuzzy sets was introduced and is explained in next section.

The spatial complexity of this QFCS is shown in Table 5.2. There is shown the number of elements of memory needed in each part of the QFCS. Thus, the addition of number of elements of memory from those parts of QFCS gives the spatial complexity that is

$$S(n, m, N, d, M) = N[2n + d^n + 2^n m M + 2] \quad (5.10)$$

In this way, the spatial complexity is $O(d^n)$ with respect to the number of input variables n of the problem. It is $O(m)$ with respect of the number of the output variables m of the problem. It is $O(N)$ with respect of the number of classifiers. And finally, it is $O(m)$ with respect of the number of fuzzy actions per output variable.

The temporal complexity of QFCS in one cycle is shown in Table 5.3. One cycle means the introduction of an input vector, activation of classifiers, updating the hyper-matrices by the learning component, application of the GA if applicable and selection of an action. In that table the number of different operations carried out by QFCS are shown. It is difficult to formulate the temporal complexity with one common operation,

	Parameter
Structure	n
	m
	$[x_l^{\min}, x_l^{\max}]$
	$[a_l^{\min}, a_l^{\max}]$
Classifiers (Cl_l)	N
	c
	d
Learning Component	β_0
	β
	γ
	δ_0
	P_E
	P_R
Discovery Component	b
	χ
	μ
	θ_{GA}

Table 5.1: Parameters of QFCS with fixed fuzzy sets.

Part of QFCS	Elements of Memory
Activation regions R^l	$N(2n)$
Prediction values $p_{d_1 \dots d_n}^l$	Nd^n
Age δ_l	N
GA application time e_l	N
Fuzzy actions	$N(2^n)(mM)$

Table 5.2: Spatial complexity of different parts of QFCS with fixed fuzzy set.

Operation	Number
Comparison with R^l	N
Finding the maximum $p_{d_1 \dots d_n}^l$	$N/c^n - 1$
Fuzzy Inference Machine Process	N/c^n
Updating of $p_{d_1 \dots d_n}^l$	1
Random selection of Cl_l	1
Application of the GA	$1/\theta_{GA}$

Table 5.3: Temporal complexity of different parts of one cycle of QFCS with fixed fuzzy set.

since they are different. Therefore, each of the operations is considered in the table as the unit to get an idea about the complexity in spite of being different. In this manner, the whole temporal complexity is the addition of all of those operations as

$$T(n, N, c, \theta_{GA}) = N + 2\frac{N}{c^n} + \frac{1}{\theta_{GA}} + 1 \quad (5.11)$$

The temporal complexity is $O(1/c^n)$ with respect of the number of input variables n . If the population is kept fixed, the complexity decreases with an increment in n but this is fictitious because, as it was said before, the population is set as $N \approx N_{AR}c^n$ where N_{AR} is the number of classifiers per activation region. Thus, the complexity is

$$T(n, N_{AR}, c, \theta_{GA}) \approx N_{AR}c^n + 2N_{AR} + \frac{1}{\theta_{GA}} + 1 \quad (5.12)$$

In this way, the temporal complexity is $O(c^n)$ with respect to the number of input variables n . It is $O(N_{AR})$ with respect of the number of classifiers per activation region N_{AR} . And it is $O(1/\theta_{GA})$ with respect to the interval time θ_{GA} of application of the GA.

5.2 QFCS with Unfixed Fuzzy Sets

QFCS with unfixed fuzzy sets [23] is a generalization of the QFCS with fixed fuzzy sets. Therefore, it works similarly to the one described before but with a few simple modifications. These modifications have something to do specifically with the activation regions of the classifiers that are not fixed any more, with the Learning Component where it is necessary to make a normalization of prediction values used by the Genetic Algorithm, and with the Genetic Algorithm that is applied now to both the activation regions and the fuzzy action parts of the fuzzy rules of the SFS_i of the classifiers. For clarity this generalization will be described with all of its details but paying special attention to the differences due to the generalization. These differences are remarked in bold.

In the generalization, each classifier contains also a small fuzzy system (SFS), a matrix containing the expected prediction, and a square sub-region of activation over the input space. The classifiers can only act over their sub-region of activation. In that sub-region of activation, each fuzzy system proposes by defuzzification a continuous vector field as an action. Therefore, when an input vector enters the QFCS, classifiers compete to place their actions according to their expected predictions. A Q-learning algorithm is used to learn the task from the environment. This algorithm is used to change the values of the matrices of the expected predictions for each classifier in the QFCS. A GA is applied to evolve rules based on their average **normalized** expected predictions. This GA evolves both **the condition and the action parts** of the fuzzy systems.

The components of the QFCS are a set of N classifiers, a performance component, a learning component, and a discovery component:

since they are different. Therefore, each of the operations is considered in the table as the unit to get an idea about the complexity in spite of being different. In this manner, the whole temporal complexity is the addition of all of those operations as

$$T(n, N, c, \theta_{GA}) = N + 2\frac{N}{c^n} + \frac{1}{\theta_{GA}} + 1 \quad (5.11)$$

The temporal complexity is $O(1/c^n)$ with respect of the number of input variables n . If the population is kept fixed, the complexity decreases with an increment in n but this is fictitious because, as it was said before, the population is set as $N \approx N_{AR}c^n$ where N_{AR} is the number of classifiers per activation region. Thus, the complexity is

$$T(n, N_{AR}, c, \theta_{GA}) \approx N_{AR}c^n + 2N_{AR} + \frac{1}{\theta_{GA}} + 1 \quad (5.12)$$

In this way, the temporal complexity is $O(c^n)$ with respect to the number of input variables n . It is $O(N_{AR})$ with respect of the number of classifiers per activation region N_{AR} . And it is $O(1/\theta_{GA})$ with respect to the interval time θ_{GA} of application of the GA.

5.2 QFCS with Unfixed Fuzzy Sets

QFCS with unfixed fuzzy sets [23] is a generalization of the QFCS with fixed fuzzy sets. Therefore, it works similarly to the one described before but with a few simple modifications. These modifications have something to do specifically with the activation regions of the classifiers that are not fixed any more, with the Learning Component where it is necessary to make a normalization of prediction values used by the Genetic Algorithm, and with the Genetic Algorithm that is applied now to both the activation regions and the fuzzy action parts of the fuzzy rules of the SFS_i of the classifiers. For clarity this generalization will be described with all of its details but paying special attention to the differences due to the generalization. These differences are remarked in bold.

In the generalization, each classifier contains also a small fuzzy system (SFS), a matrix containing the expected prediction, and a square sub-region of activation over the input space. The classifiers can only act over their sub-region of activation. In that sub-region of activation, each fuzzy system proposes by defuzzification a continuous vector field as an action. Therefore, when an input vector enters the QFCS, classifiers compete to place their actions according to their expected predictions. A Q-learning algorithm is used to learn the task from the environment. This algorithm is used to change the values of the matrices of the expected predictions for each classifier in the QFCS. A GA is applied to evolve rules based on their average **normalized** expected predictions. This GA evolves both **the condition and the action parts** of the fuzzy systems.

The components of the QFCS are a set of N classifiers, a performance component, a learning component, and a discovery component:

- **Classifiers.** Each classifier Cl_l contains a SFS $_l$ over the input $x_1 \dots x_n$ and output $a_1 \dots a_m$ spaces, and a square region R^l over that input space defined as

$$R^l = \{(x_1, \dots, x_n) \mid (\min_{x_1}^l \leq x_1 \leq \max_{x_1}^l) \wedge \dots \wedge (\min_{x_n}^l \leq x_n \leq \max_{x_n}^l)\}, \quad (5.13)$$

where $\min_{x_i}^l$ and $\max_{x_i}^l$ with $i = 1, \dots, n$ are constants, and **two sets of elements** p_{d_1, \dots, d_n}^l and N_{d_1, \dots, d_n}^l **that constitutes two dual n -dimensional matrices with the subscripts $d_i \in \{1, \dots, d\}$, $i = 1, \dots, n$ and d a constant.** The classifiers can only act over their regions R^l . **To form these regions, first in each dimension, the center and the width of the interval $[\min_{x_i}^l, \max_{x_i}^l]$ with $i = 1, \dots, n$ is calculated. The center is random in the input variable x_i . The width is random in $[c_{\min}, c_{\max}]$. Then, with the center and the width, the limits $[\min_{x_i}^l, \max_{x_i}^l]$ are obtained. These limits are truncated if they are out of the defined input space.** Each R^l is divided into d^n (d to the power of n) uniform square regions $\Delta_{d_1, \dots, d_n}^l$ that are associated one by one with p_{d_1, \dots, d_n}^l and N_{d_1, \dots, d_n}^l . Figure 5.8 shows an example. As it can be seen, everything is similar to the classifiers described in the previous section except for the regions R^l that now are not fixed regions anymore and that sub-regions $\Delta_{d_1, \dots, d_n}^l$ depend specifically on regions R^l . That is why sub-regions $\Delta_{d_1, \dots, d_n}^l$ have the super index l .

- **Performance Component.** It receives a real vector \vec{x}_0 as input; then, all classifiers that contain the input in their own regions are activated to form a match set $[M]$. At that time, the input is fuzzified and is introduced into each SFS $_l$ of the classifiers in $[M]$. The fuzzy inference machine of each SFS $_l$ works to produce an output fuzzy set that is defuzzified into an output vector $\vec{a}_l = (a_1^l, \dots, a_m^l)$ proposed by each classifier Cl_l in $[M]$. Then, one of those \vec{a}_l is selected as the output of the QFCS \vec{a}_0 in the following manner:

$$\vec{a}_0 = \arg \max_{\vec{a}_l \in [M]} [P_{d_1, \dots, d_n}], \quad (5.14)$$

where P_{d_1, \dots, d_n} is a set defined by

$$P_{d_1, \dots, d_n} = \{p_{d_1, \dots, d_n}^l \mid \vec{x}_0 \in \Delta_{d_1, \dots, d_n}^l \wedge \vec{a}_l \in [M]\}. \quad (5.15)$$

Figure 5.9 presents an example in one dimension with more detail. In this example $n = 1$ and $m = 1$. Thus, each classifier Cl_l represents a curve in the space (x_1, a_1) (Fig. 5.9.a). This curve means all of the possible responses $\vec{a}^l = [a_1^l]$ (that is a vector of one component, an scalar) from the Cl_l to all of the possible inputs $x_0 \in R^l$. Therefore, the system can be seen as a series of curves in the space (x_1, a_1) . Figure 5.9b shows the performance component described above. In that figure we can see 4 classifiers represented as red curves with their corresponding prediction vectors. The input x_0 cuts the classifiers Cl_2 and Cl_3 forming the red points. This input also cuts the prediction hyper-matrices \vec{p}^2 and \vec{p}^3 , that are vectors now, in the components p_3^2 and p_1^3 . These classifiers Cl_2 and Cl_3 propose the outputs a_1^2 and a_1^3 that compete depending of which prediction value p_3^2 and

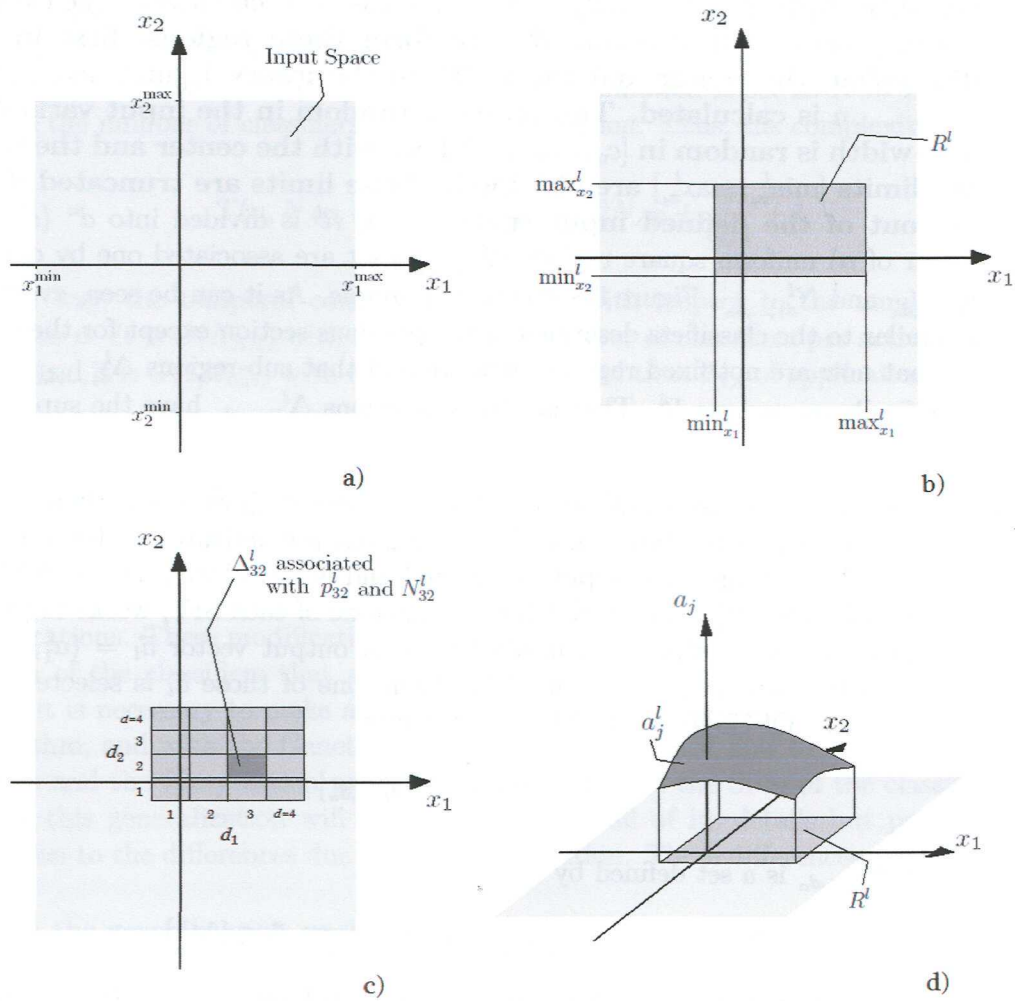


Figure 5.8: a. Input space $x_1 \dots x_n$ with $n = 2$. b. Square region R^l . c. Square sub-regions $\Delta_{d_1, \dots, d_n}^l$ with $d = 4$. d. j -th component of the vector field $\vec{a}_i = (a_1^l, \dots, a_m^l)$

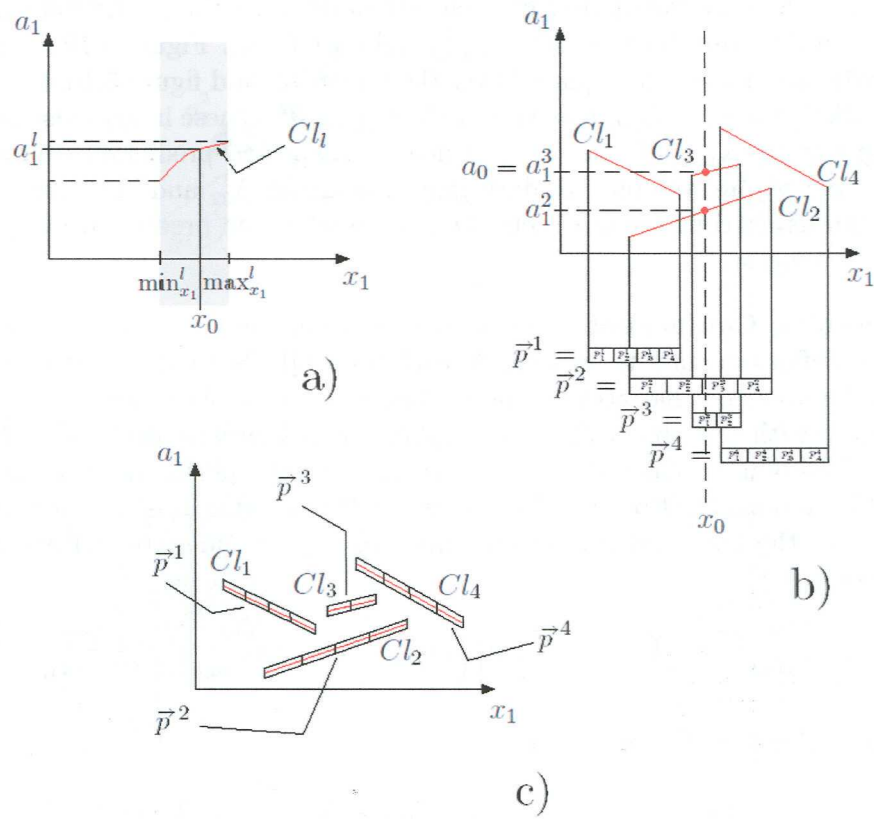


Figure 5.9: a. Meaning of a classifier. b. Performance component. c. Meaning of the prediction vectors

p_1^3 is maximal to form the output of the system a_0 . The prediction vectors can be seen as in figure 5.9c. They can be placed over the rules. It means that each value represents the expected prediction of the system over a little piece of the curve of a rule. With this way of thinking, we are representing the Q-values of the Q-function from Q-learning only over little pieces of lines, nor in the complete domain.

Figure 5.10 shows another example. In figures 5.10.a and 5.10.b are the components of the output vectors. Regions R^1 and R^2 of Cl_1 and Cl_2 intersect. The matrices of the expected prediction values p_{b_1, b_2}^l are drawn over the surfaces as grids. The gray square over each surface determines the prediction value of interest. In this case, they are for Cl_1 p_{32}^1 and for Cl_2 p_{33}^2 . Figure 5.10.c represents the prediction matrix p_{d_1, d_2}^1 placed over the region R^1 and figure 5.10.d represents the prediction matrix p_{d_1, d_2}^2 placed over the region R^2 . These images also represent the sub-regions Δ_{d_1, d_2}^i . It can be seen how the expected prediction values of interest p_{32}^1 and p_{33}^2 are the ones in dark gray and are in Δ_{32}^1 and Δ_{33}^2 respectively. The performance component selects the rule to which its prediction value, p_{32}^1 or p_{33}^2 , is the biggest.

- **Learning Component.** QFCS receives, from the environment, a reward $R(\vec{x})$ that defines the task to achieve. At each time, QFCS decides what to do exploiting the knowledge it has acquired or exploring new possible actions. The exploitation is done with probability P_E , while exploration is done with probability $P_R = 1 - P_E$. Exploitation is achieved by selecting \vec{a}_0 as in the performance component (Eq. 5.14), and exploration by selecting one of the possible $\vec{a}_i \in [M]$ at random. Each time t , the expected prediction values p_{d_1, \dots, d_n}^l at time $(t - 1)$ are adjusted as follows:

$$p_{d_1, \dots, d_n, t-1}^l \leftarrow p_{d_1, \dots, d_n, t-1}^l + \beta \left[\left(R(\vec{x}_{t-1}) + \gamma p_{d_1, \dots, d_n, t}^l \right) - p_{d_1, \dots, d_n, t-1}^l \right], \quad (5.16)$$

where $\beta, \gamma \in [0, 1]$ and

$$p_{d_1, \dots, d_n, t-1}^l = \{ p_{d_1, \dots, d_n}^l \mid \vec{x}_{t-1} \in \Delta_{d_1, \dots, d_n}^l \wedge Cl_l = Cl_{t-1} \}, \quad (5.17)$$

$$p_{d_1, \dots, d_n, t}^l = \{ p_{d_1, \dots, d_n}^l \mid \vec{x}_t \in \Delta_{d_1, \dots, d_n}^l \wedge Cl_l = Cl_t \}, \quad (5.18)$$

with \vec{x}_{t-1} and Cl_{t-1} that are the input and the classifier that was selected by the performance component at time $(t - 1)$, and \vec{x}_t and Cl_t are the input and the classifier that would be selected by exploitation at time t . β is variable through time depending on how old δ_l , the classifier Cl_l , is. In other words:

$$\beta = \begin{cases} \left[\frac{\beta_0 - 1}{\delta_0} \right] \delta_l + 1, & \text{if } \delta_l < \delta_0; \\ \beta_0, & \text{otherwise,} \end{cases} \quad (5.19)$$

where δ_0 is a constant. The age δ_l of the classifiers starts in 0 and is incremented in one unit every time step. **At each time t the expected prediction values**

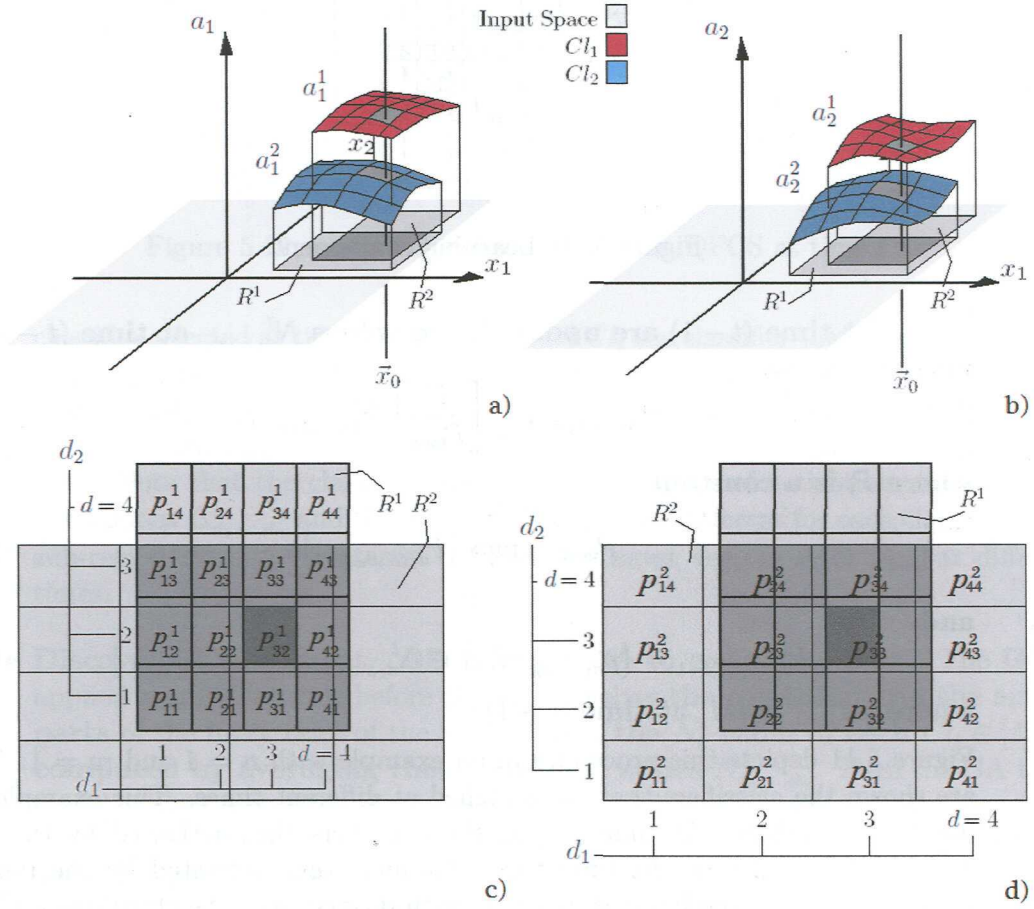


Figure 5.10: a. Component a_1^l of classifiers $Cl_l \in [M]$. b. Component a_2^l of classifiers $Cl_l \in [M]$. c. Matrix of elements p_{d_1, d_2}^1 of the classifier Cl_1 . d. Matrix of elements p_{d_1, d_2}^2 of the classifier Cl_2 .

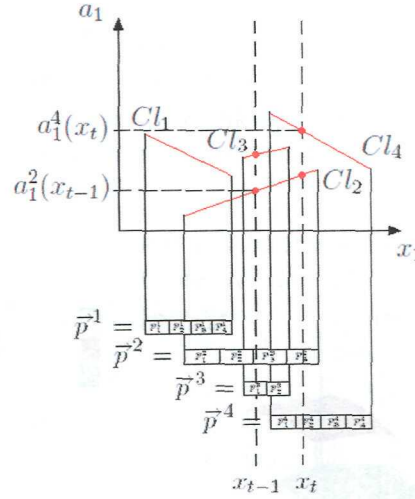


Figure 5.11: Learning component

p_{d_1, \dots, d_n}^l at time $(t-1)$ are updated, the values N_{d_1, \dots, d_n}^l at time $(t-1)$ are normalized as:

$$N_{d_1, \dots, d_n, t-1}^l = \left[\frac{P_1}{P_{\max}} \right] P_{d_1, \dots, d_n, t-1}^l, \quad (5.20)$$

where P_1 is a constant,

$$P_{\max} = \max_l [P_{d_1, \dots, d_n, t-1}^l], \quad (5.21)$$

and

$$P_{d_1, \dots, d_n, t-1}^l = \{p_{d_1, \dots, d_n}^l \mid \vec{x}_{t-1} \in \Delta_{d_1, \dots, d_n}^l \wedge Cl_l \in [M]_{t-1}\}, \quad (5.22)$$

where $[M]_{t-1}$ is $[M]$ at time $(t-1)$.

Figure 5.11 depicts this procedure in an example with $n = 1$ and $m = 1$. There are shown the classifiers that are matched at different times. This example only has four classifiers. At time $(t-1)$ the classifiers that activated by the input x_{t-1} are Cl_2 and Cl_3 . At time t the classifiers that activated by the input x_t are Cl_2 and Cl_4 . The input values cuts with dashed lines the classifiers and their prediction vectors. Those elementd cut by the dashed lines are used in the learning component. Red points represent the action of the classifiers to the input values x_{t-1} and x_t . At time $t-1$ the action selected as the action of the system is the one of the Cl_2 that is a_1^3 . At time t the action with the maximal prediction vale at time t is that of the Cl_4 that is a_1^4 . This is

$$a_1^4 = \arg \max_{a_1} [p_3^2, p_2^4] \quad (5.23)$$

Then, p_2^4 is used to modify p_3^2 that belongs to Cl_2 .

Figure 5.12 shows another example with $n = 2$ and $m = 1$. In that example the prediction value p_{14}^1 is updated using the prediction value p_{32}^5 since Cl_{t-1} , the

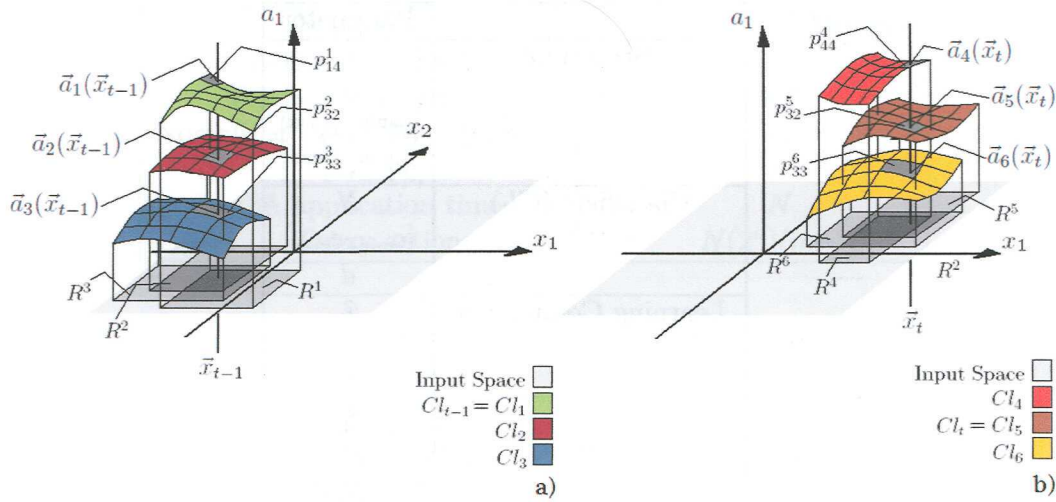


Figure 5.12: a. QFCS at time $(t - 1)$. b. QFCS at time t .

classifier selected at time $(t - 1)$, is Cl_1 and Cl_t the classifier that would be selected by the performances component (by exploitation) is Cl_5 . After this update, the prediction values p_{14}^1 , p_{32}^2 and p_{33}^3 are the $P_{d_1, \dots, d_n, t-1}^l$. In that way, if $p_{32}^2 = \max\{p_{14}^1, p_{32}^2, p_{33}^3\}$ then $P_{\max} = p_{32}^2$. With this, N_{14}^1 , N_{32}^2 and N_{33}^3 are update according to 5.20. Note that the classifiers have different regions of activation, therefore the sub-regions $\Delta_{d_1, \dots, d_n}^l$ and the matrices $p_{d_1, \dots, d_n, t}^l$ are different for each classifier. The sub-region $\Delta_{d_1, \dots, d_n}^l$ of interest contains the input vectors \vec{x}_t or \vec{x}_{t-1} at different times.

- **Discovery Component.** QFCS uses a GA to create new rules. The GA is applied over $[M]$ at time before $(t - 1)$. It evolves **the condition and the action parts** of the fuzzy rules of the SFS $_i$ s. First, **the \bar{N}_l value of each $Cl_l \in [M]$ is computed by averaging their expected values N_{d_1, \dots, d_n}^l** . Then the GA takes two classifiers from $[M]$ selecting them with **probability proportional to their \bar{N}_l** . These classifiers are copied, crossed-over with probability χ , and mutated with probability μ . One-point crossover is used. Mutation changes a 0 by a 1 or vice versa **and a real number by adding a random number with normal distribution**. Then, one of the two classifiers is inserted in $[M]$ replacing another one in $[M]$ that is **selected proportional to b/\bar{N}_l** . The GA is applied over $[M]$ on time intervals that are determined by the classifiers in $[M]$. For this, each classifier stores the last time e_l in which it was involved in a GA, so when $[M]$ happens, the average time \bar{e} of its classifiers, for the last application of the GA, is calculated by:

$$\bar{e} = \frac{\sum_{Cl_l \in [M]} (t - e_l)}{|[M]|}, \quad (5.24)$$

where $|[M]|$ represents the number of classifiers in $[M]$. If $\bar{e} \geq \theta_{GA}$ the GA is applied.

	Parameter
Structure	n
	m
	$[x_i^{\min}, x_i^{\max}]$
	$[a_i^{\min}, a_i^{\max}]$
Classifiers (Cl_i)	N
	$[c_{\min}, c_{\max}]$ d
Learning Component	β_0
	β
	γ
	δ_0
	P_E
	P_R
Discovery Component	b
	χ
	μ
	θ_{GA}

Table 5.4: Parameters of QFCS with unfixed fuzzy sets.

In this QFCS the activation regions of the classifiers that are not fixed any more. Therefore, because of the SFS is defined over activation region, the input fuzzy sets are not fixed anymore. This freedom degree is to avoid the parameter c in the version of QFCS before. Parameter c controlled the curvature formed by connecting the hyper-curves. Without fixed region of activation this parameter is not needed anymore and with that, the curvature formed by the connection among hyper-surfaces is controlled by the GA. Therefore, the GA evolves the activation regions and the fuzzy actions based on the normalized Q-values formed by the hyper-matrices. Normalized Q-values are also hyper-matrices. These new hyper-matrices are needed because now the hyper-surfaces move to those places in the state-action space where the Q-values are maximal. Thus, hyper-surfaces have the tendency to go to goal leaving holes in the state-action space. To avoid this behavior, QFCS re-scales the Q-values for all possible actions over each possible state to the maximal reward possible. This operation produces the hyper-matrices of the normalized Q-values.

The parameters to be determined are in Table 5.4. The thumb rule for the number of classifiers is followed to have about the same number of classifiers for the QFCS with the fixed fuzzy sets. Parameter $d \approx 5$ by the same reason as in the other QFCS.

The spatial complexity of this QFCS is shown in Table 5.5. There is shown the number of elements of memory needed in each part of the QFCS. Thus, the addition of number of elements of memory from those parts of QFCS gives the spatial complexity that is

$$S(n, m, N, d, M) = N[2n + 2d^n + 2^n m M + 2] \quad (5.25)$$

In this way, the spatial complexity is $O(d^n)$ with respect to the number of input variables

Part of QFCS	Elements of Memory
Activation regions R^l	$N(2n)$
Prediction values $p_{d_1 \dots d_n}^l$	Nd^n
Normalized prediction values $N_{d_1 \dots d_n}^l$	Nd^n
Age δ_l	N
GA application time e_l	N
Fuzzy actions	$N(2^n)(mM)$

Table 5.5: Spatial complexity of different parts of QFCS with unfixed fuzzy set.

Operation	Number
Comparison with R^l	N
Finding the maximum $p_{d_1 \dots d_n}^l$	$X - 1$
Fuzzy Inference Machine Process	X
Updating of $p_{d_1 \dots d_n}^l$	1
Random selection of C_l	1
Application of the GA	$1/\theta_{GA}$

Table 5.6: Temporal complexity of different parts of one cycle of QFCS with unfixed fuzzy set.

n of the problem. It is $O(m)$ with respect of the number of the output variables m of the problem. It is $O(N)$ with respect of the number of classifiers. And finally, it is $O(m)$ with respect of the number of fuzzy actions per output variable.

The temporal complexity of QFCS in one cycle is shown in Table 5.6. One cycle means the introduction of an input vector, activation of classifiers, updating the hypermatrices by the learning component, application of the GA if applicable and selection of an action. In that table is shown the number of different operations carried out by QFCS. It is also difficult to formulate the temporal complexity with one common operation, since they are different. Therefore, each of the operations are considered in the table as the unit to get an idea about the complexity in spite of being different. X is a random variable that determines the number of classifiers activated in each cycle. The density of probability of X is difficult to compute and it is not going to be obtained here. But that does not matter if the average \bar{X} is considered. In this manner, the whole temporal complexity is the average of the addition of all of those operations as

$$\bar{T}(N, X, \theta_{GA}) = N + 2\bar{X} + \frac{1}{\theta_{GA}} + 1 \quad (5.26)$$

The temporal complexity is $O(\bar{X})$ with respect of the number of classifier per activation region. It is $O(N)$ with respect of the number of classifier in the population N . And it is $O(1/\theta_{GA})$ with respect to the interval time θ_{GA} of application of the GA.

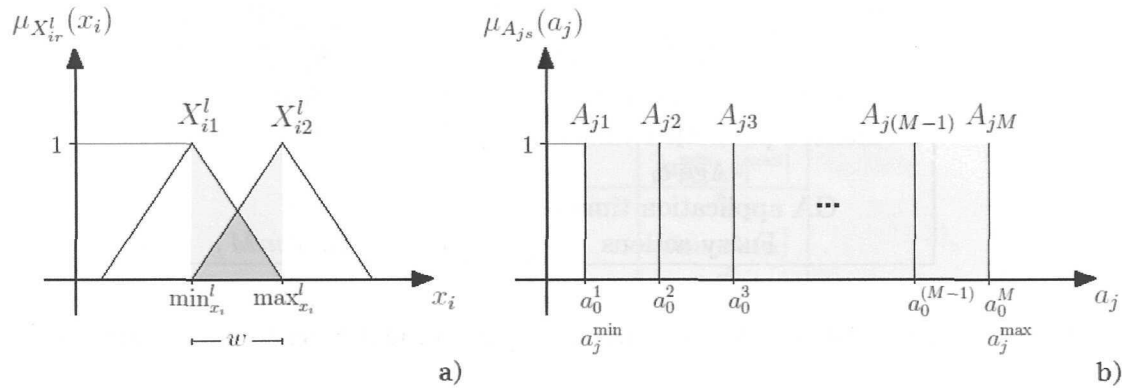


Figure 5.13: a. Memberships of the fuzzy sets from SFS_l for the input variables x_i . b. Memberships of the fuzzy sets from SFS_l for the output variables a_j .

5.3 Small Fuzzy Systems (SFS_l)

The SFS_l s of classifiers Cl_l are defined over their regions R^l and the output space. Two fuzzy sets X_{i1}^l and X_{i2}^l are defined with triangular membership functions per input linguistic variable \mathcal{X}_i and per classifier Cl_l as follows:

$$\mu_{X_{ir}^l}(x_i) = \begin{cases} \left[\frac{1}{w} \right] x_i + \left[1 - \frac{c_{ir}^l}{w} \right], & \text{if } [c_{ir}^l - w] \leq x_i < c_{ir}^l; \\ - \left[\frac{1}{w} \right] x_i + \left[1 + \frac{c_{ir}^l}{w} \right], & \text{if } c_{ir}^l \leq x_i < [c_{ir}^l + w]; \\ 0, & \text{otherwise,} \end{cases} \quad (5.27)$$

where $i \in \{1, \dots, n\}$, $r \in \{1, 2\}$, $w = (\max_{x_i}^l - \min_{x_i}^l)$, $c_{i1}^l = \min_{x_i}^l$ and $c_{i2}^l = \max_{x_i}^l$. In the output linguistic variables \mathcal{A}_j , the fuzzy sets A_{js} are singletons defined by:

$$\mu_{A_{js}}(a_j) = \begin{cases} 1, & \text{if } a_j = a_0^s; \\ 0, & \text{otherwise,} \end{cases} \quad (5.28)$$

where $j \in \{1, \dots, m\}$, $s \in \{1, \dots, M\}$ and

$$a_0^s = a_j^{\min} + (s - 1) \left[\frac{a_j^{\max} - a_j^{\min}}{M - 1} \right], \quad (5.29)$$

where a_j^{\min} and a_j^{\max} are the lower and the upper limits of a_j . Figure 5.13 depicts these fuzzy sets for the input and output variables. In this way, the fuzzy rules have the next form:

$$\text{IF } [\mathcal{X}_1 = X_1^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A}_1 = A_1^l, \dots, \mathcal{A}_m = A_m^l], \quad (5.30)$$

where $X_i^l \in \{X_{i1}^l, X_{i2}^l\}$ and A_j^l is a fuzzy disjunction (fuzzy OR) with the maximum operation (Eq. 2.5) of a subset S_j^l of the fuzzy sets in $T_j = \{A_{j1}, A_{j2}, A_{j3}, \dots, A_{j(M-1)}, A_{jM}\}$. In the starting settings each possible fuzzy set in T_j has a probability P_A of being in S_j^l . The fuzzy rules in each SFS_l use all the possible combinations of their input fuzzy sets in the condition parts, therefore, there are 2^n possible fuzzy rules in each classifier Cl_l .

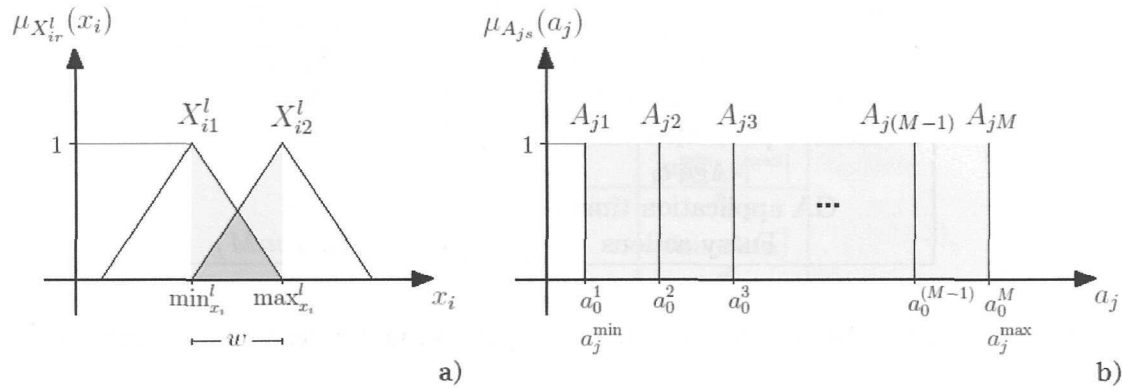


Figure 5.13: a. Memberships of the fuzzy sets from SFS_l for the input variables x_i . b. Memberships of the fuzzy sets from SFS_l for the output variables a_j .

5.3 Small Fuzzy Systems (SFS_l)

The SFS_l s of classifiers Cl_l are defined over their regions R^l and the output space. Two fuzzy sets X_{i1}^l and X_{i2}^l are defined with triangular membership functions per input linguistic variable \mathcal{X}_i and per classifier Cl_l as follows:

$$\mu_{X_{ir}^l}(x_i) = \begin{cases} \left[\frac{1}{w}\right] x_i + \left[1 - \frac{c_{ir}^l}{w}\right], & \text{if } [c_{ir}^l - w] \leq x_i < c_{ir}^l; \\ -\left[\frac{1}{w}\right] x_i + \left[1 + \frac{c_{ir}^l}{w}\right], & \text{if } c_{ir}^l \leq x_i < [c_{ir}^l + w]; \\ 0, & \text{otherwise,} \end{cases} \quad (5.27)$$

where $i \in \{1, \dots, n\}$, $r \in \{1, 2\}$, $w = (\max_{x_i}^l - \min_{x_i}^l)$, $c_{i1}^l = \min_{x_i}^l$ and $c_{i2}^l = \max_{x_i}^l$. In the output linguistic variables A_j , the fuzzy sets A_{js} are singletons defined by:

$$\mu_{A_{js}}(a_j) = \begin{cases} 1, & \text{if } a_j = a_0^s; \\ 0, & \text{otherwise,} \end{cases} \quad (5.28)$$

where $j \in \{1, \dots, m\}$, $s \in \{1, \dots, M\}$ and

$$a_0^s = a_j^{\min} + (s - 1) \left[\frac{a_j^{\max} - a_j^{\min}}{M - 1} \right], \quad (5.29)$$

where a_j^{\min} and a_j^{\max} are the lower and the upper limits of a_j . Figure 5.13 depicts these fuzzy sets for the input and output variables. In this way, the fuzzy rules have the next form:

$$\text{IF } [\mathcal{X}_1 = X_1^l \wedge \dots \wedge \mathcal{X}_n = X_n^l] \text{ THEN } [\mathcal{A}_1 = A_1^l, \dots, \mathcal{A}_m = A_m^l], \quad (5.30)$$

where $X_i^l \in \{X_{i1}^l, X_{i2}^l\}$ and A_j^l is a fuzzy disjunction (fuzzy OR) with the maximum operation (Eq. 2.5) of a subset S_j^l of the fuzzy sets in $T_j = \{A_{j1}, A_{j2}, A_{j3}, \dots, A_{j(M-1)}, A_{jM}\}$. In the starting settings each possible fuzzy set in T_j has a probability P_A of being in S_j^l . The fuzzy rules in each SFS_l use all the possible combinations of their input fuzzy sets in the condition parts, therefore, there are 2^n possible fuzzy rules in each classifier Cl_l .

Next is an example of a SFS_l in one dimension with $n = 1$, $m = 1$ and $M = 5$:

$$\begin{aligned} R_1^l &: [\mathcal{X}_1 = X_{11}^l] \rightarrow [\mathcal{A}_1 = (A_{12} \vee A_{13})], \\ R_{2^n=2}^l &: [\mathcal{X}_1 = X_{12}^l] \rightarrow [\mathcal{A}_1 = (A_{11} \vee A_{14} \vee A_{15})], \end{aligned} \quad (5.31)$$

where R_k^l means the k -th fuzzy rule of the SFS_l . And next is another example of a SFS_l in two dimensions with $n = 2$, $m = 2$ and $M = 5$:

$$\begin{aligned} R_1^l &: [\mathcal{X}_1 = X_{11}^l \wedge \mathcal{X}_2 = X_{21}^l] \rightarrow [\mathcal{A}_1 = (A_{12}), \mathcal{A}_2 = (A_{21} \vee A_{23})], \\ R_2^l &: [\mathcal{X}_1 = X_{11}^l \wedge \mathcal{X}_2 = X_{22}^l] \rightarrow [\mathcal{A}_1 = (A_{11} \vee A_{14}), \mathcal{A}_2 = (A_{25})], \\ R_3^l &: [\mathcal{X}_1 = X_{12}^l \wedge \mathcal{X}_2 = X_{21}^l] \rightarrow [\mathcal{A}_1 = (A_{14} \vee A_{15}), \mathcal{A}_2 = (A_{21} \vee A_{22})], \\ R_{2^n=4}^l &: [\mathcal{X}_1 = X_{12}^l \wedge \mathcal{X}_2 = X_{22}^l] \rightarrow [\mathcal{A}_1 = (A_{12}), \mathcal{A}_2 = (A_{24})]. \end{aligned} \quad (5.32)$$

Each SFS_l uses the Minimum Inference Engine (Eq. 2.33) [32] that uses generalized Modus Ponens Inference (Eq. 2.28). The input is fuzzified into an n -dimension singleton (Eq. 2.38). The fuzzy output is defuzzified by the center of gravity (Eq. 2.41).

Figure 5.14 shows an example in one dimension with $n = 1$ and $m = 1$. Figure 5.14.a shows how the input vector is fuzzified into a singleton membership function $\mu_{\bar{x}_0}(x_1)$ in the input space x_1 . Then, the singleton function is cylindrical expanded to the complete space $x_1 a_1$ forming a two dimension membership function $\mu_{\bar{x}_0}(x_1, a_1)$. This function is a surface. Figures 5.14.b and 5.14.c represent the membership functions in $x_1 a_1$ of the fuzzy implications R_1^l and R_2^l . These membership functions of the implications $\mu_{R_k^l}(x_1, a_1)$ are obtained by first applying cylindrical extension to the fuzzy sets of the antecedent and the consequent parts of the implications and second applying the Mamdani's Minimum operation over those antecedent and consequent. These operations mean the membership functions of the implications are obtained applying the minimum operation to the antecedent and the consequent fuzzy sets. This gives those triangles in $x_1 a_1$. Figures 5.14.d and 5.14.e show the inference operation using the Generalized Modus Ponens. The T-norm is the minimum, so with $\mu_{\bar{x}_0}(x_1, a_1)$ of the figure 5.14.a and the implications $\mu_{R_k^l}(x_1, a_1)$ of the figures 5.14.b and 5.14.c, the surfaces of the figures 5.14.d and 5.14.e are obtained. These surfaces are singletons on $x_1 a_1$. Then those singletons are projected to a_1 to obtain the results of the inference operation, the $\mu_{\bar{x}_0 \wedge R_k^l}(a_1)$ s. In figure 5.14.f, the $\mu_{\bar{x}_0 \wedge R_k^l}(a_1)$ s are combined with fuzzy union (maximum) to obtain the output fuzzy set of the SFS_l . Finally, the center of gravity is used to obtain the output vector of the SFS_l .

5.4 Implementation of Classifiers (Cl_l)

Since the classifiers have to be evolved, they need to be coded in a data structure. This data structure involves only the activation regions R^l of the classifiers Cl_l and the fuzzy consequent parts of the fuzzy rules A_j^l with $j = 1, \dots, m$. The antecedent parts of the fuzzy rules are not involved since they are fixed membership functions over regions R^l , therefore they never change explicitly. They only change implicitly when activation regions R^l do. The elements placed in the data structure for a classifier Cl_l in one

Next is an example of a SFS_l in one dimension with $n = 1$, $m = 1$ and $M = 5$:

$$\begin{aligned} R_1^l &: [\mathcal{X}_1 = X_{11}^l] \rightarrow [\mathcal{A}_1 = (A_{12} \vee A_{13})], \\ R_{2^n=2}^l &: [\mathcal{X}_1 = X_{12}^l] \rightarrow [\mathcal{A}_1 = (A_{11} \vee A_{14} \vee A_{15})], \end{aligned} \quad (5.31)$$

where R_k^l means the k -th fuzzy rule of the SFS_l . And next is another example of a SFS_l in two dimensions with $n = 2$, $m = 2$ and $M = 5$:

$$\begin{aligned} R_1^l &: [\mathcal{X}_1 = X_{11}^l \wedge \mathcal{X}_2 = X_{21}^l] \rightarrow [\mathcal{A}_1 = (A_{12}), \mathcal{A}_2 = (A_{21} \vee A_{23})], \\ R_2^l &: [\mathcal{X}_1 = X_{11}^l \wedge \mathcal{X}_2 = X_{22}^l] \rightarrow [\mathcal{A}_1 = (A_{11} \vee A_{14}), \mathcal{A}_2 = (A_{25})], \\ R_3^l &: [\mathcal{X}_1 = X_{12}^l \wedge \mathcal{X}_2 = X_{21}^l] \rightarrow [\mathcal{A}_1 = (A_{14} \vee A_{15}), \mathcal{A}_2 = (A_{21} \vee A_{22})], \\ R_{2^n=4}^l &: [\mathcal{X}_1 = X_{12}^l \wedge \mathcal{X}_2 = X_{22}^l] \rightarrow [\mathcal{A}_1 = (A_{12}), \mathcal{A}_2 = (A_{24})]. \end{aligned} \quad (5.32)$$

Each SFS_l uses the Minimum Inference Engine (Eq. 2.33) [32] that uses generalized Modus Ponens Inference (Eq. 2.28). The input is fuzzified into an n -dimension singleton (Eq. 2.38). The fuzzy output is defuzzified by the center of gravity (Eq. 2.41).

Figure 5.14 shows an example in one dimension with $n = 1$ and $m = 1$. Figure 5.14.a shows how the input vector is fuzzified into a singleton membership function $\mu_{\bar{x}_0}(x_1)$ in the input space x_1 . Then, the singleton function is cylindrical expanded to the complete space $x_1 a_1$ forming a two dimension membership function $\mu_{\bar{x}_0}(x_1, a_1)$. This function is a surface. Figures 5.14.b and 5.14.c represent the membership functions in $x_1 a_1$ of the fuzzy implications R_1^l and R_2^l . These membership functions of the implications $\mu_{R_k^l}(x_1, a_1)$ are obtained by first applying cylindrical extension to the fuzzy sets of the antecedent and the consequent parts of the implications and second applying the Mamdani's Minimum operation over those antecedent and consequent. These operations mean the membership functions of the implications are obtained applying the minimum operation to the antecedent and the consequent fuzzy sets. This gives those triangles in $x_1 a_1$. Figures 5.14.d and 5.14.e show the inference operation using the Generalized Modus Ponens. The T-norm is the minimum, so with $\mu_{\bar{x}_0}(x_1, a_1)$ of the figure 5.14.a and the implications $\mu_{R_k^l}(x_1, a_1)$ of the figures 5.14.b and 5.14.c, the surfaces of the figures 5.14.d and 5.14.e are obtained. These surfaces are singletons on $x_1 a_1$. Then those singletons are projected to a_1 to obtain the results of the inference operation, the $\mu_{\bar{x}_0 \wedge R_k^l}(a_1)$ s. In figure 5.14.f, the $\mu_{\bar{x}_0 \wedge R_k^l}(a_1)$ s are combined with fuzzy union (maximum) to obtain the output fuzzy set of the SFS_l . Finally, the center of gravity is used to obtain the output vector of the SFS_l .

5.4 Implementation of Classifiers (Cl_l)

Since the classifiers have to be evolved, they need to be coded in a data structure. This data structure involves only the activation regions R^l of the classifiers Cl_l and the fuzzy consequent parts of the fuzzy rules A_j^l with $j = 1, \dots, m$. The antecedent parts of the fuzzy rules are not involved since they are fixed membership functions over regions R^l , therefore they never change explicitly. They only change implicitly when activation regions R^l do. The elements placed in the data structure for a classifier Cl_l in one

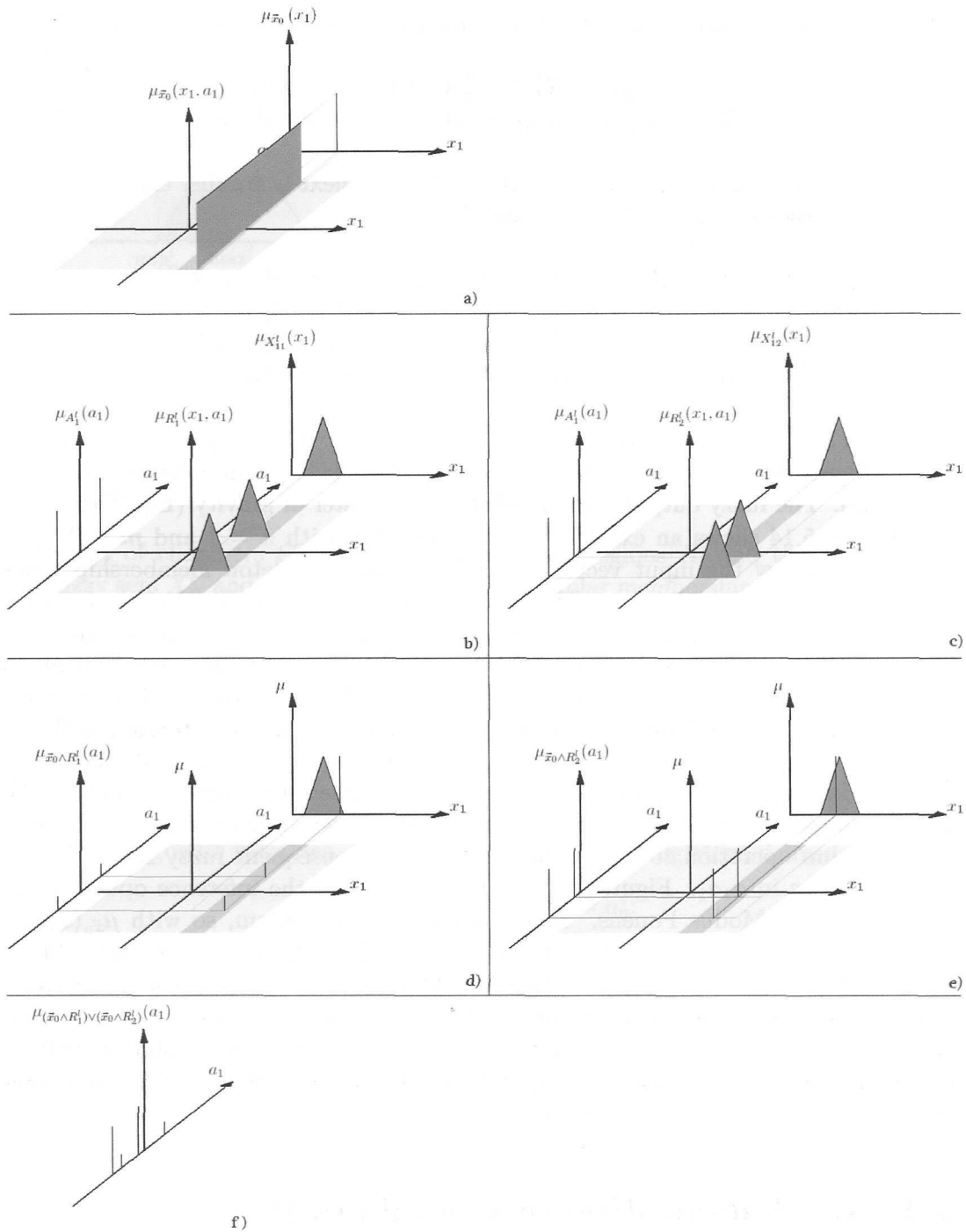


Figure 5.14: a. Fuzzyfication and cylindrical extension of the input vector \vec{x}_0 . b. Membership function of the implication R_1^l . c. Membership function of the implication R_2^l . d. Membership function of the inference operation with fuzzy rule R_1^l . e. Membership function of the inference operation with fuzzy rule R_2^l . f. Membership function of the union over the results of the inference operations.

dimension are:

Fuzzy Rule	R^l		A_{11}	A_{12}	...	A_{1M}
R_1^l :	$\min_{x_1}^l$	$\max_{x_1}^l$	0	1	...	1
R_2^l :	$\min_{x_1}^l$	$\max_{x_1}^l$	1	0	...	0

(5.33)

where the 1s represent the fuzzy sets in S_1^l . As R^l is the same in all the fuzzy rules, it is placed once in the data structure. The sets S_1^l of each fuzzy rules are placed one after the other. Therefore the data structure is:

R^l		S_1^l of R_1^l	S_1^l of R_2^l
$\min_{x_1}^l$	$\max_{x_1}^l$	01...1	10...0

(5.34)

In this way, a structure for a classifier in two dimensions would be:

R^l				S_1^l of R_1^l	S_1^l of R_2^l	S_1^l of R_3^l	S_1^l of R_4^l
$\min_{x_1}^l$	$\max_{x_1}^l$	$\min_{x_2}^l$	$\max_{x_2}^l$	01...1	10...0	01...0	11...0

S_2^l of R_1^l	S_2^l of R_2^l	S_2^l of R_3^l	S_2^l of R_4^l
11...1	00...1	10...0	01...0

(5.35)

The Genetic Algorithm is given these data structures of the classifiers.

5.5 General Discussion

QFCS was designed to solve continuous problems by reward. These continuous problems are the ones defined by the n -Environment Problem. This problem is defined over a set of n continuous input variables known as the states space and with another set of m continuous output variables known as the action space. Actions define transitions in state space that depend on the particular problem. The objective of the problem is to reach by means of the actions a small region called goal where a reward is given.

QFCS with fixed fuzzy sets intended to learn the Q-function in the combined state-action space but only in those regions determined by the classifiers. To do this, classifiers represent relationships between the states and the actions. Classifiers do so through the use of SFSs. A hyper-matrix in each classifier approximates the Q-function.

The main difference in letting the activation regions evolve is that classifiers let the GA that task of finding the curvature of the solution, which is very important because in QFCS with the fixed fuzzy sets this is a parameter that has to be adjusted. Finding the correct curvature of the solution introduces more complexity to the GA and this is going to be seen in the experiments, where in general there is not a better performance.

dimension are:

Fuzzy Rule	R^l		A_{11}	A_{12}	\dots	A_{1M}
R_1^l :	$\min_{x_1}^l$	$\max_{x_1}^l$	0	1	\dots	1
R_2^l :	$\min_{x_1}^l$	$\max_{x_1}^l$	1	0	\dots	0

(5.33)

where the 1s represent the fuzzy sets in S_1^l . As R^l is the same in all the fuzzy rules, it is placed once in the data structure. The sets S_1^l of each fuzzy rules are placed one after the other. Therefore the data structure is:

R^l		S_1^l of R_1^l	S_1^l of R_2^l
$\min_{x_1}^l$	$\max_{x_1}^l$	01...1	10...0

(5.34)

In this way, a structure for a classifier in two dimensions would be:

R^l				S_1^l of R_1^l	S_1^l of R_2^l	S_1^l of R_3^l	S_1^l of R_4^l
$\min_{x_1}^l$	$\max_{x_1}^l$	$\min_{x_2}^l$	$\max_{x_2}^l$	01...1	10...0	01...0	11...0

S_2^l of R_1^l	S_2^l of R_2^l	S_2^l of R_3^l	S_2^l of R_4^l
11...1	00...1	10...0	01...0

(5.35)

The Genetic Algorithm is given these data structures of the classifiers.

5.5 General Discussion

QFCS was designed to solve continuous problems by reward. These continuous problems are the ones defined by the n -Environment Problem. This problem is defined over a set of n continuous input variables known as the states space and with another set of m continuous output variables known as the action space. Actions define transitions in state space that depend on the particular problem. The objective of the problem is to reach by means of the actions a small region called goal where a reward is given.

QFCS with fixed fuzzy sets intended to learn the Q-function in the combined state-action space but only in those regions determined by the classifiers. To do this, classifiers represent relationships between the states and the actions. Classifiers do so through the use of SFSs. A hyper-matrix in each classifier approximates the Q-function.

The main difference in letting the activation regions evolve is that classifiers let the GA that task of finding the curvature of the solution, which is very important because in QFCS with the fixed fuzzy sets this is a parameter that has to be adjusted. Finding the correct curvature of the solution introduces more complexity to the GA and this is going to be seen in the experiments, where in general there is not a better performance.

5.6 Summary

There are two QFCSs: one with fixed fuzzy sets as input and the other one with unfixed fuzzy sets. One is the generalization of the other one. QFCS with the fixed fuzzy sets is compounded by:

- **SET OF CLASSIFIERS.** There exists a fixed set of classifiers. Each classifier contains an activation region, a SFS defined over that activation region and a hyper-matrix.
- **PERFORMANCE COMPONENT.** It determines how QFCS works. It receives an input vector. Classifiers that contain that input vector in their activation regions form a match set. Classifiers in the match set compete to place their action. The action is calculated by the SFS. The selected classifier is the one that has the maximum Q-value.
- **LEARNING COMPONENT.** It uses Q-learning. This algorithm modifies the hyper-matrices of classifiers that work as approximations to the Q-function. The learning is applied in the match set in the previous time.
- **DISCOVERY COMPONENT.** It uses a GA to evolve the fuzzy actions of the classifiers.

QFCS with the unfixed fuzzy sets has the same components as QFCS with fixed fuzzy sets but with some modifications. The modifications are in the classifiers, in the learning component and in the discovery component.

- **SET OF CLASSIFIERS.** Each classifier introduces another hyper-matrix that defines a normalization of the Q-values.
- **LEARNING COMPONENT.** At the same time that the hyper-matrix of the Q-values in each classifier is modified, the normalized hyper-matrix has to be normalized again.
- **DISCOVERY COMPONENT.** The GA evolves the activation regions and the fuzzy actions of the classifiers.

Chapter 6

Test Problems

This chapter describes the problems that were used to test QFCS. These problems are the Frog Problem proposed in [36] and five different instances of the n -Environment Problem proposed in [22]. The first was used to prove that QFCS is able to deal with it since it is a simple version of the more general 1-Environment. This is a one-step problem since the frog acts once and receives a reward that depends only on its present state and action. Therefore, Q-learning is not required. The second is an abstract problem that QFCS is able to deal with. It matches many different real problems where continuous state spaces are needed. QFCS was tested in five different instances of this problem: World_a^{1D} , World_a^{2D} , World_b^{2D} , Particle_a^{1D} and Particle_b^{1D} . Each instance introduces a new level of complexity. Therefore, World_a^{1D} is a navigation task in one continuous dimension with displacement equal to its actions that are continuous. The problem is to reach the goal. World_a^{2D} and World_b^{2D} are navigation tasks in two dimensions with displacement equal to their vector actions which are also continuous. In World_a^{2D} there are no obstacles while in World_b^{2D} there is one. This introduces difficulty because the system has to avoid the obstacles. Particle_a^{1D} and Particle_b^{1D} simulates an inertial particle in one dimension. The task is to take a particle that initiates with velocity zero to another position with velocity zero too. These particle problems are more complex than the others before because of the inertia. The dynamic of an inertial particle is described at the end of the chapter.

6.1 The Frog Problem

The Frog Problem [36, 28] consists of a frog that lives in a continuous one-dimensional space $x \in [x^{\min}, x^{\max}] = [0, 2]$, and that can jump a distance $a \in [a^{\min}, a^{\max}] = [0, 1]$. A fly is placed on $x = 1$ and the frog is placed at a random position $x \in [0, 1]$. The goal for the frog is to jump just once and catch the fly. That is considered a trial. After each trial, the frog receives a reward given by:

$$R(x, a) = \begin{cases} x + a, & \text{if } (x + a) \leq 1; \\ 2 - (x + a), & \text{otherwise.} \end{cases} \quad (6.1)$$

Figure 6.1.a shows the reward function. An optimal solution means to catch the fly always from each possible position of the frog. Thus, the optimal solution of the problem

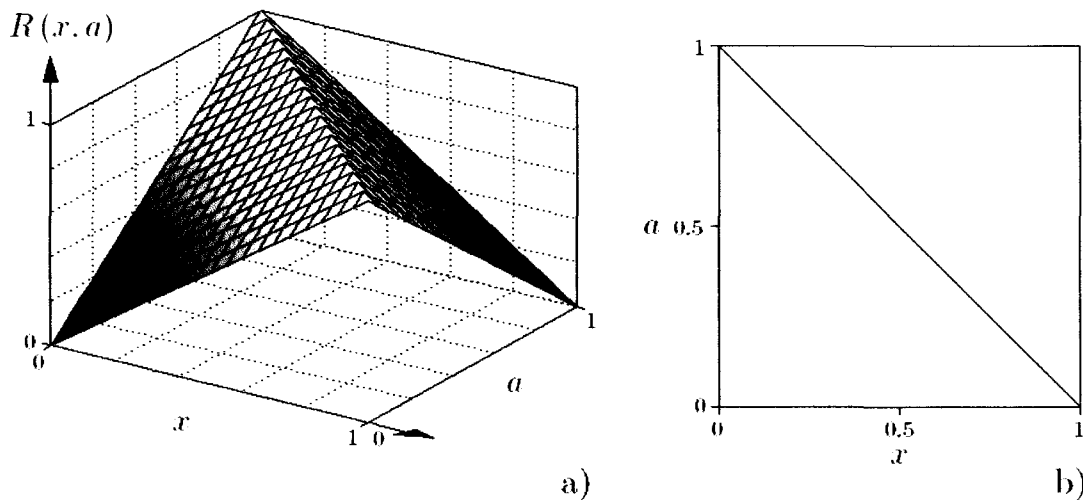


Figure 6.1: a. Reward function $R(x, a)$. b. Optimal solution

is shown in Fig. 6.1.b. That figure represents the action the frog takes in each position. For example, if the frog is in position 0.2 the action that represents the length of the jump has to be 0.8 to catch the fly. If the jump were 0.79 the frog would not reach the fly. The frog really has a set of continuous actions that define the length of the jump. The action is whichever value in the range $[0, 1]$, but it is a one-step problem since the frog only has one chance. This problem is tackled to demonstrate that QFCS can solve it. This problem is important because it is the first with continuous actions in the LCS's literature. This problem is difficult because the classifiers have to represent all possible actions which is impossible because they are infinite due to continuity.

Kovacs et al. [36] solved this problem by implementing three different architectures that used a combination of two XCSFs. The approaches were: one was based on interpolation, the second on an actor-critic paradigm, and the third on treating the action as another input for the system. The first uses one $XCSF_1$ with a discretization of the actions of the problem and the other $XCSF_2$ is programmed to learn a continuous function based on the actions taken by the $XCSF_1$. Thus, this system works as an interpolation algorithm. The second uses a system $XCSF_1$ that makes an action based on a weight and the input vectors, and an $XCSF_2$ that works as an approximation function to predict the reward to be obtained. In the third approximation the input and output of the problem are introduced as input to XCSF. This system approaches the solution by pieces of continuous curves. QFCS works better in this problem than the approximations mentioned before; moreover, QFCS can deal with problems that are more complex than the frog problem. A generalized frog problem could be the same frog in one dimension but with a fly too far to catch in one jump. It means that to catch the fly it has to make more than one jump. This problem could be taken to the frog in two dimensions that is also more complex. QFCS can deal with these more complex problems of the fly.

Trung et al. [28] also solve this frog problem modifying XCSF by the introduction of two GAs. First, a change in the representation of classifiers was done. This

change introduced the calculus of an action through a dot product of two vectors: an action vector and an input vector. This generates lines in the input-output space. The prediction values now were approximated by a plane over the state-action space and were used to extrapolate the prediction of reward out of the curve that represented by the dot product of the action vector and input vector. Therefore, one GA evolves the action vectors of the classifiers and the other the classifiers. This approach works better than the ones proposed by Kovacs et al. [36] and it works better than QFCS but this algorithm only was tested on this simple problem.

Next section defines a more complex problem than the frog problem called the n -Environment Problem. The frog problem is an instance of the n -Environment Problem. Therefore, this problem is a more general one.

6.2 The n -Environment Problem

The n -Environment Problem [22] is an n -dimensional continuous space that is determined by a square region:

$$R = \{(x_1, \dots, x_n) \mid (x_1^{\min} \leq x_1 \leq x_1^{\max}) \wedge \dots \wedge (x_n^{\min} \leq x_n \leq x_n^{\max})\}, \quad (6.2)$$

where x_i^{\min} and x_i^{\max} represent the lower and the upper limits of the variable x_i . An agent lives in R taking the position \vec{x} . This environment has a set of m -dimensional continuous action vectors defined as:

$$\vec{a} = \{(a_1, \dots, a_m) \mid (a_1^{\min} \leq a_1 \leq a_1^{\max}) \wedge \dots \wedge (a_m^{\min} \leq a_m \leq a_m^{\max})\}, \quad (6.3)$$

where a_j^{\min} and a_j^{\max} represent the lower and the upper limits of the component a_j of \vec{a} . The goal in the environment is defined as a sub-region $R_G \subset R$. There can be obstacles O_k that are also defined as sub-regions $R_{O_k} \subset R$. Obstacles are prohibited regions. The reward function is defined as:

$$R(\vec{x}) = \begin{cases} p_I, & \text{if } \vec{x} \in R_G; \\ 0, & \text{otherwise,} \end{cases} \quad (6.4)$$

and is given to the agent each time step. A trial in this environment means to begin in a random position $\vec{x} \in R$ and finish in the goal R_G . Since a trial takes more than one step, the n -Environment Problem is a multistep problem.

As it was said, the n -Environment problem is defined as an abstract problem. It has n continuous input variables that define continuous vector states and m continuous output variables that define continuous vector actions. The transition function can be whichever. One important characteristic is that a reward is obtained only when reaching the goal. This is important because it introduces a greater degree of difficulty, contrary to what happens with the frog problem, where the frog receives reward even if it does not catch the fly. Giving reward only in goal states resembles that QFCS has to find out how to solve the task. If QFCS never reaches the goal, it will never know how to solve the task. As an example, let us suppose that a dog is left on a street in the city. It is required that the dog learn how to reach a certain house and to do so the dog is given

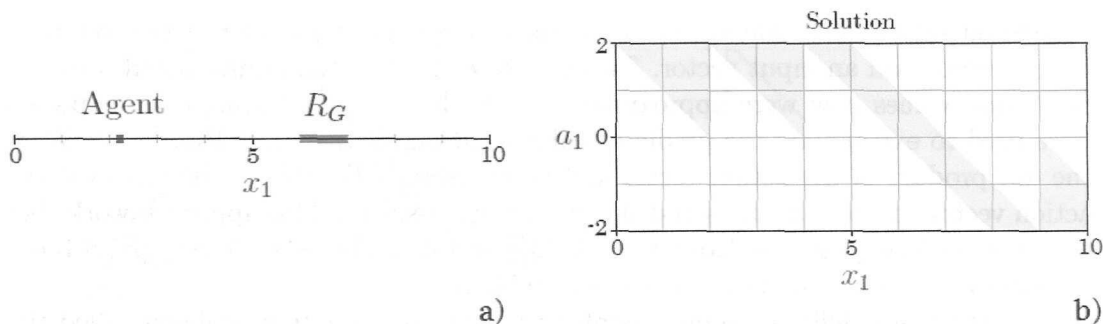


Figure 6.2: a. World_a^{1D} . b. Optimal solution of World_a^{1D} .

a big piece of steak when it reaches the house. Each time the dog reaches the house, it is left in some another street at random. This procedure is repeated many times again until the dog knows how to reach the house.

Thus, the n -Environment problem can match many different real problems that are defined in state spaces and that the solution means to reach some states, known as goals. Examples of those are the navigation tasks, controlling the motion of an inertial particle, controlling temperature, etc. Some of these problems were used to test QFCS, all of them with different levels of difficulty. These instances are described in the following sections.

6.2.1 The 1-Environment Problem: World_a^{1D}

World_a^{1D} is a navigation task in one continuous dimension. It has $n = 1$ and $m = 1$. The limits of the space R are

$$R = \{x_1 \mid (0 \leq x_1 \leq 10)\}. \quad (6.5)$$

The limits of the possible actions \vec{a} are

$$\vec{a} = \{a_1 \mid (-2 \leq a_1 \leq 2)\}. \quad (6.6)$$

An action a_1 means a displacement, so the next position x_1 of the agent that is in position x_1 is

$$x_1' = x_1 + a_1. \quad (6.7)$$

There is only one goal defined as

$$R_G = \{x_1 \mid (6 \leq x_1 \leq 7)\}. \quad (6.8)$$

There are no obstacles. In the reward function, $p_I = 10$. This problem can be thought as a Generalized Frog Problem, where the frog can jump forward and backward more than once to catch the fly. This is particularly important because if the fly is too far away, it cannot be reached by the frog in a single step. Figure 6.2.a shows World_a^{1D} . A solution is optimal when the goal is reached with the smaller number of steps from each position. Therefore, the problem has many optimal solutions. Figure 6.2.b shows

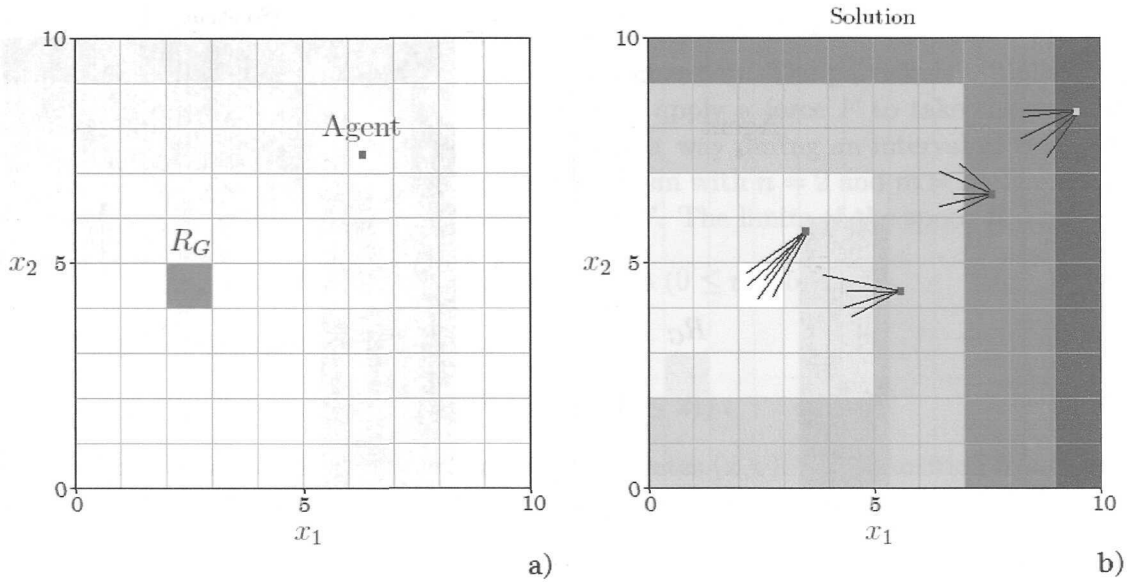


Figure 6.3: a. World_a^{2D} . b. Optimal solution of World_a^{2D} .

in gray the optimal solution on space $x_1 a_1$. This solution is not a function but a region. For example, if the agent is in position 2 the optimal possible actions are whichever in $[0,1]$. It means that taking 0.5 or 0.9 or 0.001 are optimal actions. They are optimal actions because whichever takes the agent to the goal in a smaller number of steps.

6.2.2 The 2-Environment Problem: World_a^{2D}

World_a^{2D} is a navigation task in two continuous dimension without obstacles. It has $n = 2$ and $m = 2$. The limits of the space R are

$$R = \{(x_1, x_2) \mid (0 \leq x_1 \leq 10) \wedge (0 \leq x_2 \leq 10)\}. \quad (6.9)$$

The limits of the possible actions \vec{a} are

$$\vec{a} = \{(a_1, a_2) \mid (-2 \leq a_1 \leq 2) \wedge (-2 \leq a_2 \leq 2)\}. \quad (6.10)$$

An action \vec{a} means a displacement, so the next position \vec{x}' of the agent that is in position \vec{x} is

$$\vec{x}' = \vec{x} + \vec{a}. \quad (6.11)$$

These actions allow the agent move in each possible direction. In the reward function $R(\vec{x})$, $p_I = 10$. There are no obstacles and there is only one goal defined as

$$R_G = \{(x_1, x_2) \mid (2 \leq x_1 \leq 3) \wedge (4 \leq x_2 \leq 5)\}. \quad (6.12)$$

This problem can be also thought as a Generalized Frog Problem in two dimensions. Figure 6.3.a shows World_a^{2D} . A solution is optimal when the goal is reached with the smaller number of steps from each position. Therefore, this problem also has many

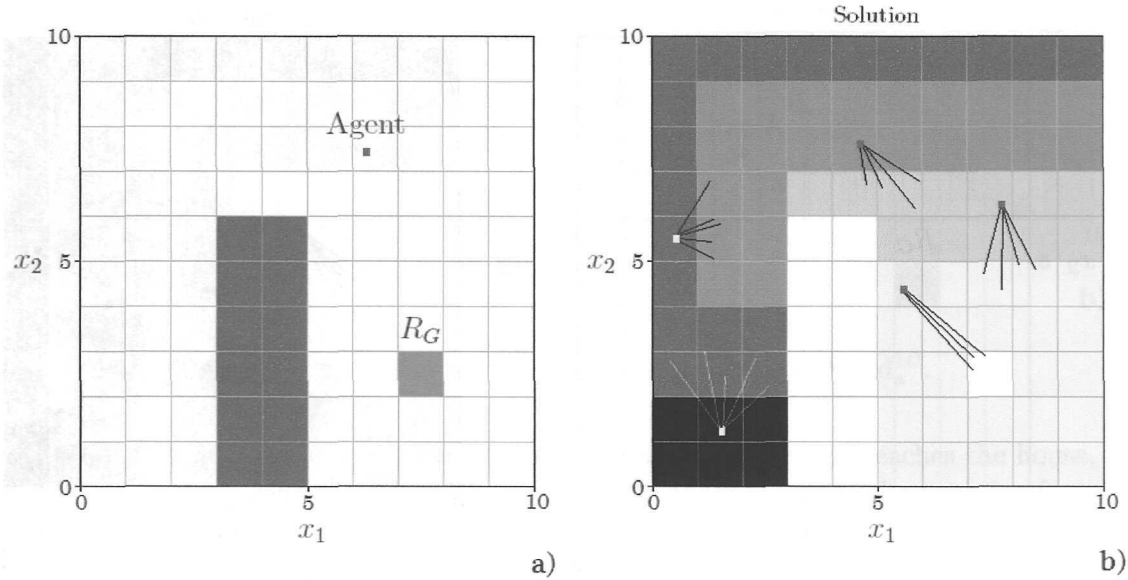


Figure 6.4: a. World_b^{2D} . b. Optimal solution of World_b^{2D} .

optimal solutions. Figure 6.3.b shows, in different gray levels, the regions that have points that can be connected to reach the goal in an optimal path. The width of these regions depends on both, the possible actions defined by Eq. 6.10, and the transitions defined by Eq. 6.11. These regions are consecutive. Some points are shown with their optimal transitions. Since there are many transitions for each point, the solution is also a region. In this way, it is considered that the agent can reach the goal using the smaller number of possible steps but with a trajectory that does not have the minimal distance. Q-learning does not take into account the minimization of this distance while QFCS does, in spite of using Q-learning. This problem is more difficult than the navigation task in one dimension since the dimensionality is higher.

6.2.3 The 2-Environment Problem: World_b^{2D}

World_b^{2D} is similar to World_a^{2D} except for the definition of the goal that has an obstacle. The goal is defined as

$$R_G = \{(x_1, x_2) \mid (7 \leq x_1 \leq 8) \wedge (2 \leq x_2 \leq 3)\}, \quad (6.13)$$

and the obstacle is defined as

$$R_{O_1} = \{(x_1, x_2) \mid (3 \leq x_1 \leq 5) \wedge (0 \leq x_2 \leq 6)\}. \quad (6.14)$$

Figure 6.4 shows World_b^{2D} and its optimal solution similarly to World_a^{2D} . This problem introduces another level of difficulty due to the existence of an obstacle. This means that the system has to avoid the obstacle to reach the goal.

6.2.4 The 2-Environment Problem: Particle_a^{1D}

Particle_a^{1D} simulates an inertial particle in one dimension. The particle is initialized in a position x with velocity $v = 0$. The goal is to apply a force F to take the particle to the goal R_G . This force is applied in a constant way during an interval of time Δt . Therefore, this problem is a 2-Environment Problem with $n = 2$ and $m = 1$. The input space is $x_1x_2 = xv$ and the output space is $a_1 = F$. The limits of the space R are

$$R = \{(x, v) \mid (0 \leq x \leq 10) \wedge (0 \leq v \leq 5)\}. \quad (6.15)$$

The limits of the possible actions \vec{a} are

$$\vec{a} = \{F \mid (-2 \leq F \leq 2)\}. \quad (6.16)$$

The force F determines the transitions between states (x, v) . The transition from state (x, v) to state (x', v') applying the force F in a constant way during the time interval Δt is given by

$$x' = x + \left[\frac{F}{2m} \right] \Delta t^2 + v\Delta t; \quad (6.17)$$

$$v' = v + \left[\frac{F}{m} \right] \Delta t. \quad (6.18)$$

that are the Eqs. 6.50 and 6.51. This transition function is more complicated than the ones for the navigation tasks. The main complexity is introduced by the inertia of the particle. This inertia hinders the agent to move in any arbitrary direction in the state space xv . Therefore, there are defined directions that are preferred by the particle depending on which velocity it has.

The mass of the particle is $m = 1$ for convenience. In the reward function $R(x, v)$, $p_I = 10$. There are no obstacles and there is only one goal defined as

$$R_G = \{(x, v) \mid (9 \leq x \leq 10) \wedge (0 \leq v \leq 0.25)\}. \quad (6.19)$$

As, it can be seen, this problem is similar to a car that can only accelerate forward and reduce its velocity; negative velocities are not allowed. Therefore, applying the brakes does not matter if the car is stopped, i.e., $v = 0$.

Figure 6.5.a shows Particle_a^{1D}. An optimal solution also means to reach the goal with the smaller number of steps. Figure 6.5.b shows the optimal solution. The space xv is divided in two regions. The maximal force $F = 2$ is applied on region in light gray to accelerate the particle to its maximum possible. The minimal force $F = -2$ is applied on the region in dark gray to desaccelerate the particle until stop. The curve that divides both regions can be determined as follows. From relations of figure 6.5.b, if the initial position of the particle is x , then the particle has to accelerate its maximum possible applying the maximum force possible F_M until it reaches half the distance to goal

$$x' = x + \Delta x; \quad (6.20)$$

$$= x + \left[\frac{x_G - x}{2} \right]. \quad (6.21)$$

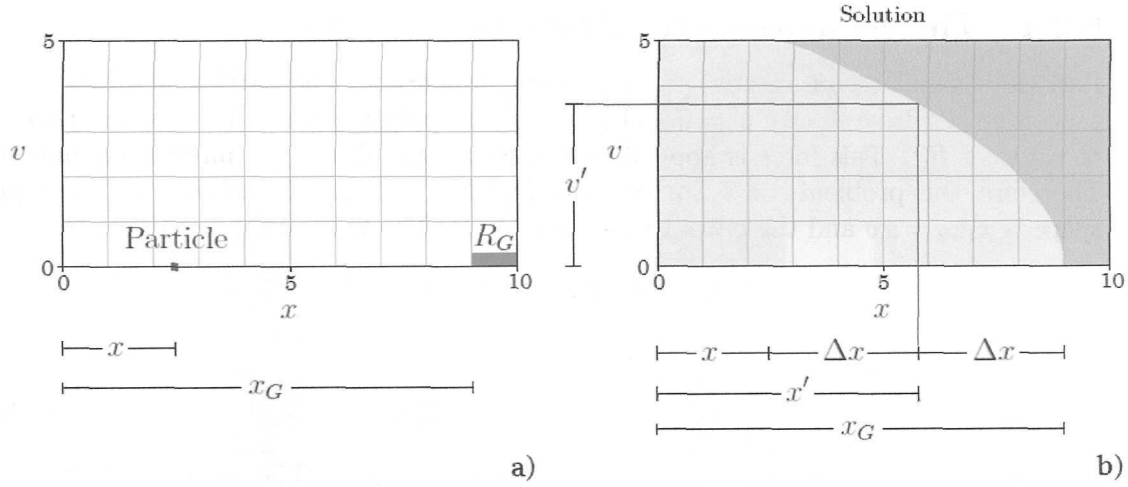


Figure 6.5: a. Particle_a^{1D}. b. Optimal solution of Particle_a^{1D}.

Eq. 6.17 with the replacement of Eq. 6.21, the initial velocity $v = 0$ and the maximal force F_M gives

$$x + \left[\frac{x_G - x}{2} \right] = x + \left[\frac{(F_M)}{2m} \right] \Delta t^2 + (0)\Delta t; \quad (6.22)$$

$$\left[\frac{x_G - x}{2} \right] = \left[\frac{F_M}{2m} \right] \Delta t^2; \quad (6.23)$$

$$\left[\frac{2m}{F_M} \right] \left[\frac{x_G - x}{2} \right] = \Delta t^2; \quad (6.24)$$

$$\sqrt{\left[\frac{m}{F_M} \right]} [x_G - x] = \Delta t. \quad (6.25)$$

Replacing Δt , $v = 0$ and F_M in Eq. 6.18 it is possible to find the final velocity

$$v' = (0) + \left[\frac{(F_M)}{m} \right] \left(\sqrt{\left[\frac{m}{F_M} \right]} [x_G - x] \right); \quad (6.26)$$

$$= \sqrt{\left[\frac{F_M}{m} \right]} [x_G - x]. \quad (6.27)$$

Getting x from Eq. 6.21

$$x' = x + \left[\frac{x_G - x}{2} \right]; \quad (6.28)$$

$$= x + \frac{x_G}{2} - \frac{x}{2}; \quad (6.29)$$

$$= \frac{x}{2} + \frac{x_G}{2}; \quad (6.30)$$

$$2x' = x + x_G; \quad (6.31)$$

$$2x' - x_G = x, \quad (6.32)$$

and replacing it in Eq. 6.27

$$v' = \sqrt{\left[\frac{F_M}{m}\right] [x_G - (2x' - x_G)]}; \quad (6.33)$$

$$= \sqrt{\left[\frac{F_M}{m}\right] [x_G - 2x' + x_G]}; \quad (6.34)$$

$$= \sqrt{\left[\frac{F_M}{m}\right] [2x_G - 2x']}; \quad (6.35)$$

$$= \sqrt{\left[\frac{2F_M}{m}\right] [x_G - x']}. \quad (6.36)$$

Eq. 6.36 is in general the curve of points (x', v') that separates the regions. The particular curve of Particle_a^{1D} with the replacement of the quantities $x_G = 9$, $m = 1$ and $F_M = 2$ is

$$v' = \sqrt{4[9 - x']}. \quad (6.37)$$

This problem is introduced with the spirit of taking QFCS to be applied in more real problems.

6.2.5 The 2-Environment Problem: Particle_b^{1D}

Particle_b^{1D} is similar to Particle_a^{1D} except for the definition of the space and goal. Therefore, the limits of the space R are

$$R = \{(x, v) \mid (0 \leq x \leq 10) \wedge (-3 \leq v \leq 3)\}, \quad (6.38)$$

and the goal is defined as

$$R_G = \{(x, v) \mid (4.5 \leq x \leq 5.5) \wedge (-0.25 \leq v \leq 0.25)\}. \quad (6.39)$$

In this problem, the particle can have negative velocities. Figure 6.6 shows Particle_b^{1D} and the optimal solution applying the maximal force possible similarly to Particle_a^{1D}. This problem is more complex than Particle_a^{1D}. The difficulty is introduced by allowing negative velocities.

6.3 Dynamics of an Inertial Particle

Let us have a particle of mass m , in position x_0 with velocity v_0 at time t_0 . Then, let us apply on that particle a constant force F during a time interval Δt . This is shown in figure 6.7. The solution $x(t)$ of the movement of the particle is determined by the differential equation

$$F = m \frac{d^2 x}{dt^2}, \quad (6.40)$$

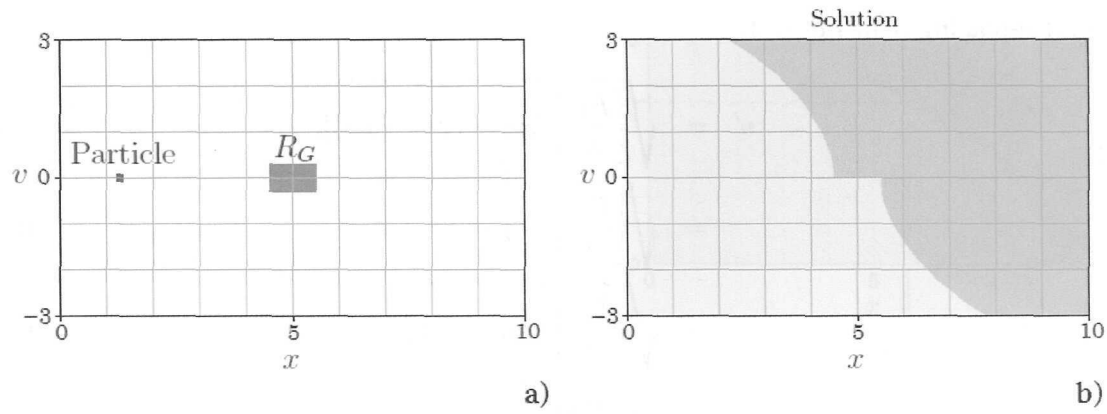


Figure 6.6: a. Particle^{1D}. b. Optimal solution of Particle^{1D}.

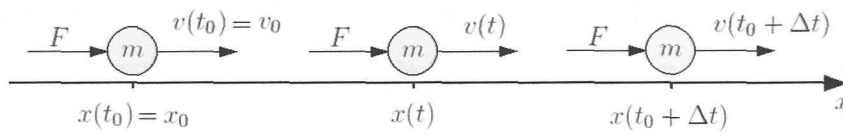


Figure 6.7: a. Initial conditions of the particle with mass m .

under the initial conditions $x(t_0) = x_0$ and $v(t_0) = v_0$. Eq. 6.40 is known as the Newton's Movement Equation. Since F is constant during time interval Δt , Eq. 6.40 can be directly integrated to get

$$Ft + C_1 = m \frac{dx}{dt}, \quad (6.41)$$

where C_1 is a constant that can be determined by the initial conditions as

$$C_1 = [mv_0 - Ft_0]. \quad (6.42)$$

Then Eq. 6.41 takes the form

$$Ft + [mv_0 - Ft_0] = m \frac{dx}{dt}, \quad (6.43)$$

that can be directly integrated one more time to give

$$\frac{1}{2}Ft^2 + [mv_0 - Ft_0]t + C_2 = mx, \quad (6.44)$$

where C_2 is also another constant that can be determined by the initial conditions as

$$C_2 = mx_0 - \frac{1}{2}Ft_0^2 - [mv_0 - Ft_0]t_0. \quad (6.45)$$

In that way, the solution $x(t)$ is given by

$$x(t) = \frac{1}{m} \left[\frac{1}{2} F t^2 + [m v_0 - F t_0] t + \left[m x_0 - \frac{1}{2} F t_0^2 - [m v_0 - F t_0] t_0 \right] \right]; \quad (6.46)$$

$$= \frac{1}{m} \left[\frac{1}{2} F t^2 + [m v_0 - F t_0] t + \left[m [x_0 - v_0 t_0] - F t_0^2 \left[\frac{1}{2} - 1 \right] \right] \right]; \quad (6.47)$$

$$= \left[\frac{F}{2m} \right] t^2 + \left[\frac{m v_0 - F t_0}{m} \right] t + \left[x_0 - v_0 t_0 + \frac{F t_0^2}{2m} \right], \quad (6.48)$$

and by deriving Eq. 6.48, the velocity $v(t)$ is obtained as

$$v(t) = \left[\frac{F}{m} \right] t + \left[v_0 - \frac{F t_0}{m} \right]. \quad (6.49)$$

Thus, the particle at time $t = (t_0 + \Delta t)$ will be in position $x(t_0 + \Delta t)$ and velocity $v(t_0 + \Delta t)$. From Eqs. 6.48 and 6.49, and with some simple algebra, the position and velocity will be

$$x(t_0 + \Delta t) = x_0 + \left[\frac{F}{2m} \right] \Delta t^2 + v_0 \Delta t; \quad (6.50)$$

$$v(t_0 + \Delta t) = v_0 + \left[\frac{F}{m} \right] \Delta t. \quad (6.51)$$

6.4 General Discussion

The problems defined in this section are the ones used to test QFCS. The main characteristic of these problems is that they have a set of continuous vectors as actions. It introduces difficulty for traditional LCSs and, in general, for all Artificial Intelligence. LCSs relate one action to many possible states of the problem, regardless of whether the state are discrete or continuous. It is not so difficult to modify classifiers to work with continuous inputs. XCSF is an example of this. The problem is that it is difficult to imagine or conceive a LCS that manages at the same time a set of continuous actions. The number of actions would be infinite because of the continuity. If it had to associate one action to many states per each classifier, it would take an infinite number of classifiers. Q-learning discretized the possible actions, but when the discretization is higher the algorithm does not converge. Therefore, a mechanism is needed to represent those infinite number of actions.

Kovacs et al. [36] and Trung et al. [28] took small steps in this direction. Their representations introduce continuous relationships between the states and actions. But those are not enough since those work only in one-step problems.

The n -Environment problem introduces more complexity since it has continuous vectors as actions and it is multi-step. But one more characteristic that the frog problem does not have is that of receiving reward only in goal states. The frog problem receives reward even in those states that are not part of the goal. This is called continuous reward.

6.5 Summary

QFCS was designed to deal with the n -Environment problem. The n -Environment problem is defined in a continuous state space with a continuous action space. The task is defined by reaching some goal. The goal is some region defined in the state space. The transition function is arbitrary and defines how the states changed by the performed actions. A reward is given when the goal is reached. QFCS was tested in five different instances of this problem with different levels of difficulty. Next are the instances used:

- WORLD_a^{1D} . It is a navigation task in one continuous dimension with continuous displacements.
- WORLD_a^{2D} . It is a navigation task in two continuous dimensions with continuous vector displacements. It does not have an obstacle.
- WORLD_b^{2D} . It is a navigation task in two continuous dimensions with continuous vector displacements. It has an obstacle.
- PARTICLE_a^{1D} . It is a simulation of an inertial particle with only positive velocities.
- PARTICLE_b^{1D} . It is a simulation of an inertial particle with positive and negative velocities.

QFCS was also tested on the frog problem. This is about a frog that has to catch a fly in a continuous environment with continuous jumps.

Chapter 7

Experiments and Results

This chapter describes how the experiments with QFCS were conducted, the employed methodology, the parameter settings, and the results. These experiments were conducted to test the capabilities of QFCS. Therefore, the problems defined in Chapter 6 have different levels of complexity. The experiments begin with the frog problem [36] because, as it was said before, this problem is the most tackled in the literature that has the characteristic of a set of continuous actions. The experiments continue with the five instances of the n -Environment problem: World_a^{1D} , World_a^{2D} , World_b^{2D} , Particle_a^{1D} and Particle_b^{1D} . These instances have something to do with navigation task in one or two dimensions and with the motion of an inertial particle. The methodology that was used is similar to the ones that are found in LCSs literature. In literature, this methodology consists of reporting averaged results. The important variables to measure are the number of steps the system takes to reach the goal during learning, this measures the convergences of learning, the action function learned by the system that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal. The next sections give a whole description of the used methodology, the experiment settings and the obtained results.

7.1 Parameters and Experiment Structure

In all of the experiments, there are some parameters that are general because they were set up to the same values. It means that they took the same values in all of the problems. In the classifiers structure $c = 4$ and $d = 5$. These values were selected considering the characteristics of QFCS they control. c controls the curvature of the action function while d controls how well-defined is the approximation of Q-function. A value of 4 of c makes the curvature not being too high and not being flat at all. A high value of d makes classifiers take too much time to learn the Q-function, but the approximation is very good. A low value of d approximates the Q-function with very small precision. A value of 5 of d makes the robustness of the approximation of the Q-learning acceptable. These values resulted to be acceptable in the experiments. In the SFSs, $M = 5$ and $P_A = 0.05$. These parameters defined the number of output fuzzy sets of the SFSs and the probability of being in the fuzzy rules. The value of M was selected based on not having too many output fuzzy sets and, at the same time, not a

Chapter 7

Experiments and Results

This chapter describes how the experiments with QFCS were conducted, the employed methodology, the parameter settings, and the results. These experiments were conducted to test the capabilities of QFCS. Therefore, the problems defined in Chapter 6 have different levels of complexity. The experiments begin with the frog problem [36] because, as it was said before, this problem is the most tackled in the literature that has the characteristic of a set of continuous actions. The experiments continue with the five instances of the n -Environment problem: World_a^{1D} , World_a^{2D} , World_b^{2D} , Particle_a^{1D} and Particle_b^{1D} . These instances have something to do with navigation task in one or two dimensions and with the motion of an inertial particle. The methodology that was used is similar to the ones that are found in LCSs literature. In literature, this methodology consists of reporting averaged results. The important variables to measure are the number of steps the system takes to reach the goal during learning, this measures the convergences of learning, the action function learned by the system that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal. The next sections give a whole description of the used methodology, the experiment settings and the obtained results.

7.1 Parameters and Experiment Structure

In all of the experiments, there are some parameters that are general because they were set up to the same values. It means that they took the same values in all of the problems. In the classifiers structure $c = 4$ and $d = 5$. These values were selected considering the characteristics of QFCS they control. c controls the curvature of the action function while d controls how well-defined is the approximation of Q-function. A value of 4 of c makes the curvature not being too high and not being flat at all. A high value of d makes classifiers take too much time to learn the Q-function, but the approximation is very good. A low value of d approximates the Q-function with very small precision. A value of 5 of d makes the robustness of the approximation of the Q-learning acceptable. These values resulted to be acceptable in the experiments. In the SFSs, $M = 5$ and $P_A = 0.05$. These parameters defined the number of output fuzzy sets of the SFSs and the probability of being in the fuzzy rules. The value of M was selected based on not having too many output fuzzy sets and, at the same time, not a

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Classifiers (Cl_i)	c	4	-	-
	d	5	5	-
Learning Component	β_0	0.2	0.2	-
	β	-	-	0.2
	γ	0.1	0.1	0.1
	P_E	0.7	0.7	0.7
	P_R	0.3	0.3	0.3
Discovery Component	b	0.5	0.5	-
	χ	0.8	0.8	-
	μ	0.04	0.04	-
SFS _{<i>i</i>s}	M	5	5	-
	P_A	0.05	0.05	-

Table 7.1: General parameters in all of the experiments.

few ones. The selection was for robustness. P_A was obtained by experimentation. It was desired to have in the initial conditions of QFCS classifiers uniformly distributed over the state-action space of the problem. Therefore, a value of 0.05 accomplishes that objective. The next parameters were set according to the ones used in other LCSs in literature. Thus, in the learning component $\beta_0 = 0.2$, $\beta = 0.2$, $\gamma = 0.1$, $P_E = 0.7$ and $P_R = 0.3$. In the discovery component $b = 0.5$, $\chi = 0.8$ and $\mu = 0.04$. These values are also shown in Table 7.1. The rest of the parameters $[x_i^{\min}, x_i^{\max}]$, $[a_j^{\min}, a_j^{\max}]$, $[c_{\min}, c_{\max}]$, θ_{GA} , N , n , m , and δ_0 depend on the particular characteristics of the problem and the type of QFCS used (the one with the fixed fuzzy sets or the one with the unfixed fuzzy sets).

The experimentation was done following the standard methodology used in literature similar to Lanzi et al. [15]. This methodology consists of reporting averaged results comparing with Q-learning. The averages are taken over samplings of size 10. The important variables to measure are the number of steps the system makes to reach the goal during learning, measuring the learning convergence, the action function learned by the system that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal.

Therefore, the followed methodology was to report averages over samplings of size 20. All of the results obtained with QFCS were compared with the ones obtained by Q-learning with a high discretization. An instance of the sampling is an agent that uses QFCS or Q-learning with initial conditions defined at random. In QFCS, the initial conditions are the initial values of the hyper-matrices and the output fuzzy sets of the fuzzy rules in the SFSs. In Q-learning, the initial conditions are the initial Q-values of the Q-function. The experiments consist of runs of the 20 agents in learning stage called learning runs. During a learning run, an agent is allowed a number of trials N_T in the problem. Each trial consists of leaving the agent in some random state of the problem and letting it act until it reaches the goal. Thus, each trial consists of a maximal number

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	1	1	1
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 1]	[0, 1]	[0, 1]
	$[a_1^{\min}, a_1^{\max}]$	[0, 1]	[0, 1]	[0, 1]
	Δx_1	-	-	0.01
	Δa_1	-	-	0.01
Classifiers (Cl_i)	N	200	200	-
	$[c_{\min}, c_{\max}]$	-	[0.1, 0.4]	-
Learning Component	δ_0	500	500	-
Discovery Component	θ_{GA}	1000	3000	-
Experiments	N_T	500000	500000	500000
	N_S	1	1	1

Table 7.2: Particular parameters in the Frog Problem.

of steps N_S or less if QFCS reaches the goal before the N_S steps. If this maximal number of steps N_S is reached the trial finishes. The important variables to measure are the number of steps the system makes to reach the goal during learning that measures the convergences in learning, the action function learned by the system that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal.

7.2 The Frog Problem

The frog problem, as it was defined before, is about a frog that lives in a line. The objective is that the frog has to jump just once to catch a fly. This problem is single-step and has continuous reward because the frog is given a reward regardless of catching the fly or not. The particular parameters used in the Frog Problem for QFCS with fixed fuzzy sets, QFCS with unfixed fuzzy sets and Q-learning are shown in Table 7.2. Each axes of the space xa in Q-learning is discretized over 100 intervals. Therefore, $\Delta x = 0.01$ and $\Delta a = 0.01$. This is so, because the problem is defined in continuous variables. The size of the population is 200 which gives about 50 classifiers per activation region since there are 4. N_S is 1 since the problem is single step. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning.

At the start of each experiment, the p'_{d_1} in QFCSs and the Q-values of Q-learning are set up at random between [0, 10]. 10 is used because it is the maximum value obtained by reward. Thus, Figure 7.1 shows the initial states of three instances of this problem. Figure 7.1.a is an instance of QFCS with fixed fuzzy sets. Figure 7.1.b is an instance of QFCS with unfixed fuzzy sets. And Figure 7.1.c is an instance of Q-learning. In that figure, Fig. 7.1, rows represent the same concept —rules, Q-values, normalized Q-values and actions— but with different systems. Therefore, the classifiers are in the

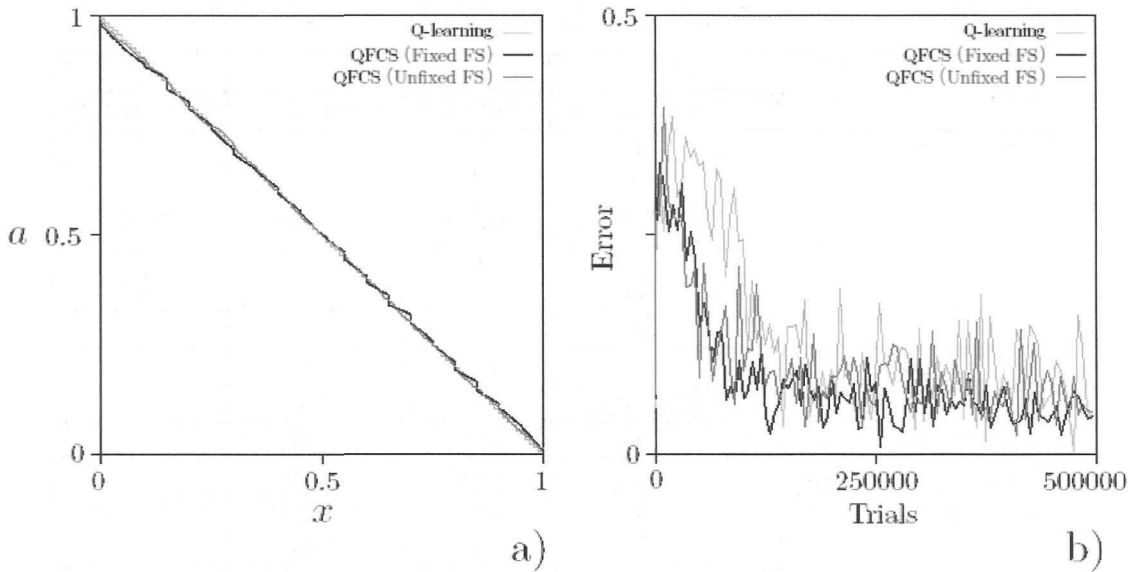


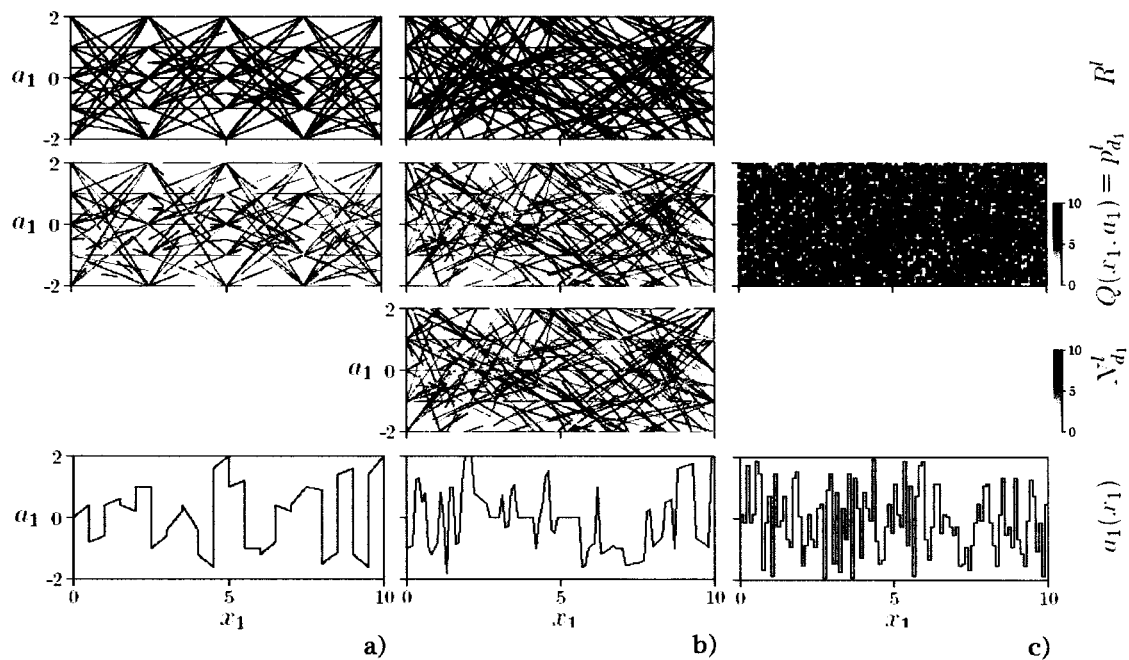
Figure 7.5: a. Average over 20 runs of the action learned in the Frog Problem. b. Average over 20 runs of the error in the Frog Problem.

These results show how QFCS can solve the frog problem and that was the objective because of the importance of that problem in LCSs literature. In comparison, QFCS and Q-learning have a similar performance. It means they converge similar and with almost the same results. But it is important to take into account that QFCS learns the action function by parts that are continuous while Q-learning learns the discretized action. On the other hand, QFCS is expected to model more complex problems where a high discretization is needed and Q-learning would not converge. The next section describes results with some instances of the n -Environment Problem.

7.3 The 1-Environment Problem: $World_a^{1D}$

$World_a^{1D}$ is an instance of the 1-Environment problem. It is a navigation task in one continuous dimension. The objective is from some position in the environment to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous actions. The particular parameters used for QFCS with fixed fuzzy sets, QFCS with unfixed fuzzy sets and Q-learning are presented in Table 7.3. The axes of space xa in Q-learning are also discretized over 100 intervals. Therefore, $\Delta x = 0.01$ and $\Delta a = 0.01$. This is so because the problem is defined over continuous variables. The size of the population is 200 which gives about 50 classifiers per activation region since there are 4. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. When each experiment begins, the $p_{d_1}^l$ in QFCSs and the Q-values of Q-learning are also set up at random between $[0, 10]$. 10 is used because it is the maximum value obtained by reward. Thus, Figure 7.6 shows the initial states of three instances of this problem. Similarly to the Frog Problem, Fig.

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	1	1	1
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[a_1^{\min}, a_1^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	Δx_1	-	-	0.1
	Δa_1	-	-	0.1
Classifiers (Cl_i)	N	200	200	-
	$[c_{\min}, c_{\max}]$	-	[1, 4]	-
Learning Component	δ_0	500	500	-
Discovery Component	θ_{GA}	1000	3000	-
Experiments	N_T	200000	200000	200000
	N_S	∞	100	∞

Table 7.3: Particular parameters in World_a^{1D} .Figure 7.6: a. Initial State of the QFCS with fixed fuzzy sets in World_a^{1D} . b. Initial State of the QFCS with unfixed fuzzy sets in World_a^{1D} . c. Initial State of Q-learning in World_a^{1D} .

7.6.a is an instance of QFCS with fixed fuzzy sets, Fig. 7.6.b is an instance of QFCS with unfixed fuzzy sets, and Fig. 7.6.c is an instance of Q-learning. In that figure, Fig. 7.6, rows represent the same variable but with different systems. Therefore, the classifiers are in the first row, the prediction values or Q-values in a gray scale are in the second row, the normalized prediction values in a gray scale in the third row and the action functions in the last row. Since this problem is also one-dimensional, the classifiers represent curves over x_1a_1 , too. These curves are obtained by drawing, for each classifier and for each possible position in the activation region of the classifier, its proposed action. Thus, for each position, the classifier gives an action and these values, the position and the action, are represented using a point in the space xa . This is done for all of the positions in the activation region of the classifier producing a curve. This process is repeated for all of the classifiers in the population generating the curves shown in the first row. The curves represented by the classifiers are also almost uniformly distributed over space x_1a_1 . QFCS with the fixed fuzzy sets has also the curves represented by the classifiers distributed over four regions that are

$$\begin{aligned}
 \text{Region}_1 &= \{(x, a) \mid (0 \leq x \leq 2.5) \wedge (-2 \leq a \leq 2)\}, \\
 \text{Region}_2 &= \{(x, a) \mid (2.5 \leq x \leq 5) \wedge (-2 \leq a \leq 2)\}, \\
 \text{Region}_3 &= \{(x, a) \mid (5 \leq x \leq 7.5) \wedge (-2 \leq a \leq 2)\}, \\
 \text{Region}_4 &= \{(x, a) \mid (7.5 \leq x \leq 10) \wedge (-2 \leq a \leq 2)\};
 \end{aligned} \tag{7.2}$$

while the QFCS with unfixed fuzzy set does not. This is due to the activation regions. QFCS represents the Q-function of Q-learning only in those curves determined by the classifiers. Therefore, the Q-function is not represented in all of the space x_1a_1 as Q-learning demands. Moreover, it is expected that QFCS will be able to evolve classifiers to find out those regions of the space x_1a_1 in the form of curves such that the Q-learning is determinant for making decisions. The initial proposed action by any of these systems has no meaning.

Figure 7.7 shows 3 instances of an experiment of 20 QFCSs with fixed fuzzy sets. The first row in that figure shows how classifiers evolved in such a way to represent the Q-function over the region where the Q-values are maximal given an x_1 . It means that the classifiers follow the optimal solution region. For example, from Fig. 7.7, if $x = 2.5$ then classifiers are arranged in such a way that they cover the action in the range $[0.5, 2]$. Classifiers do not cover all of this region but some points of it. For example point $(2.5, 0)$ is not covered. The curvature of the solution is controlled by the parameter c . The optimal solution of this problem is a region, it means that there are many possible optimal solutions and QFCS finds one. The solutions found by QFCS are in the third row. These are curves. Contrary of what happened with the frog problem where the optimal solution was a line, here the solutions show some curvatures. It is easy to see that QFCS could modify the classifiers to follow the curvature of the optimal solutions. The second row shows the prediction values p'_{a_1} . This prediction values are a function of the space x_1a_1 . Therefore, the values are shown in a gray scale. Black is for 10 and white for 0. These gray colors are over the curves represented by the classifiers. These learned Q-values show that the Q-function has its maximum values where the curves of the classifiers are. The third row shows the learned actions. This action function is continuous with some discontinuities and is in the region of the optimal solution

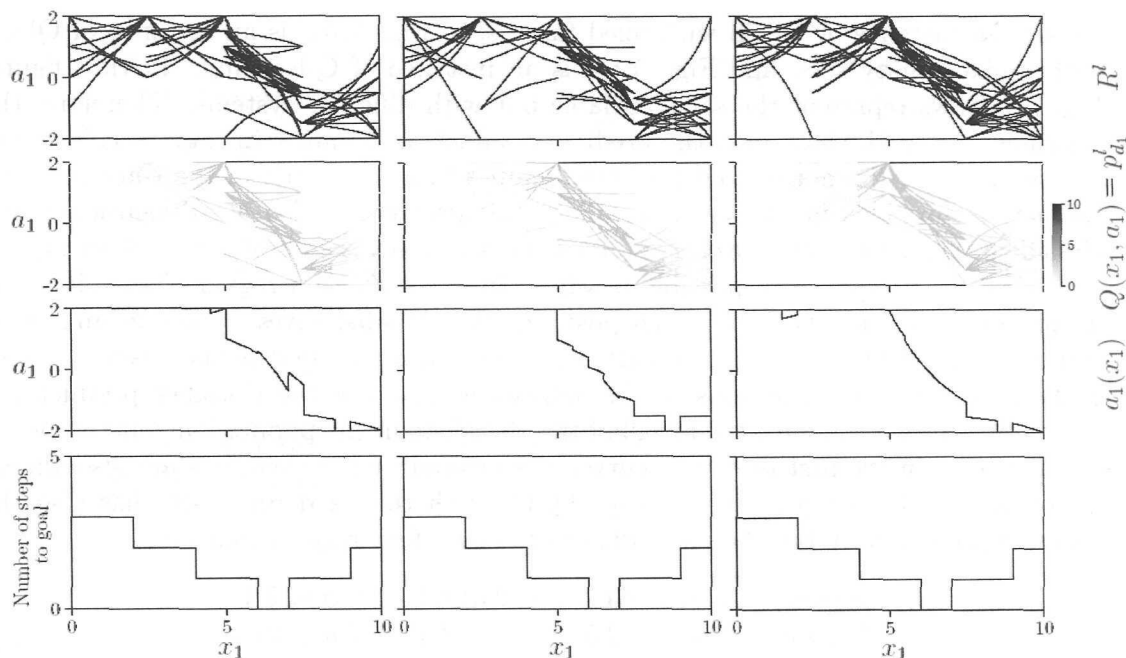


Figure 7.7: Three instances of the QFCS with fixed fuzzy sets in World_a^{1D} .

according to Fig. 6.2.b except for those places where it is difficult for QFCS to evolve classifiers since the fuzzy sets are fixed. This condition makes the curves be on a region in $[\min_{x_1}^l, \max_{x_1}^l]$ with their ends in $x = \min_{x_1}^l$ and $x = \max_{x_1}^l$. These ends can only move over lines $x = \min_{x_1}^l$ and $x = \max_{x_1}^l$. In this case, the difficulty is in points $(0, 2)$, $(2, 2)$, $(4, 2)$ and $(9, -2)$, where the solution is only one point and not a region (Fig. 6.2.b). Therefore, these functions are sub-optimal solutions. The prediction values $p_{d_1}^l$ are maximal only in those places where the action is important and the evolved classifiers are over those places too. The fourth row shows the number of steps the system takes to reach the goal from each possible position x . QFCS does not do more than three steps. Therefore, QFCS can learn an optimal solution.

Similar results are shown in figure 7.8 where there are 3 instances of an experiment of 20 QFCSs with unfixed fuzzy sets. The classifiers were evolved similarly to QFCS with the fixed fuzzy sets, but in this case, there is no limitation to some activation regions. Classifiers could change in any direction without any restrictions. Therefore, the curvature of the solution is followed better than with the fixed activation regions. This is why the liberations of the shape of the activation regions was done. Again, the prediction values $p_{d_1}^l$ have higher values in places closer to the goal and over the curves represented by classifiers. The normalized prediction values $N_{d_1}^l$ are shown in the third row. They are always in a gray scale similar to Q-values. These values were introduced in QFCS with unfixed fuzzy sets to allow the GA evolve classifiers over the solution action curve. Thus, they have to reach, per each possible state, the maximum value of reward 10. Thus, the normalized function is bigger close to the solution action curve. Classifiers are better distributed in this QFCS that in the other one. The action

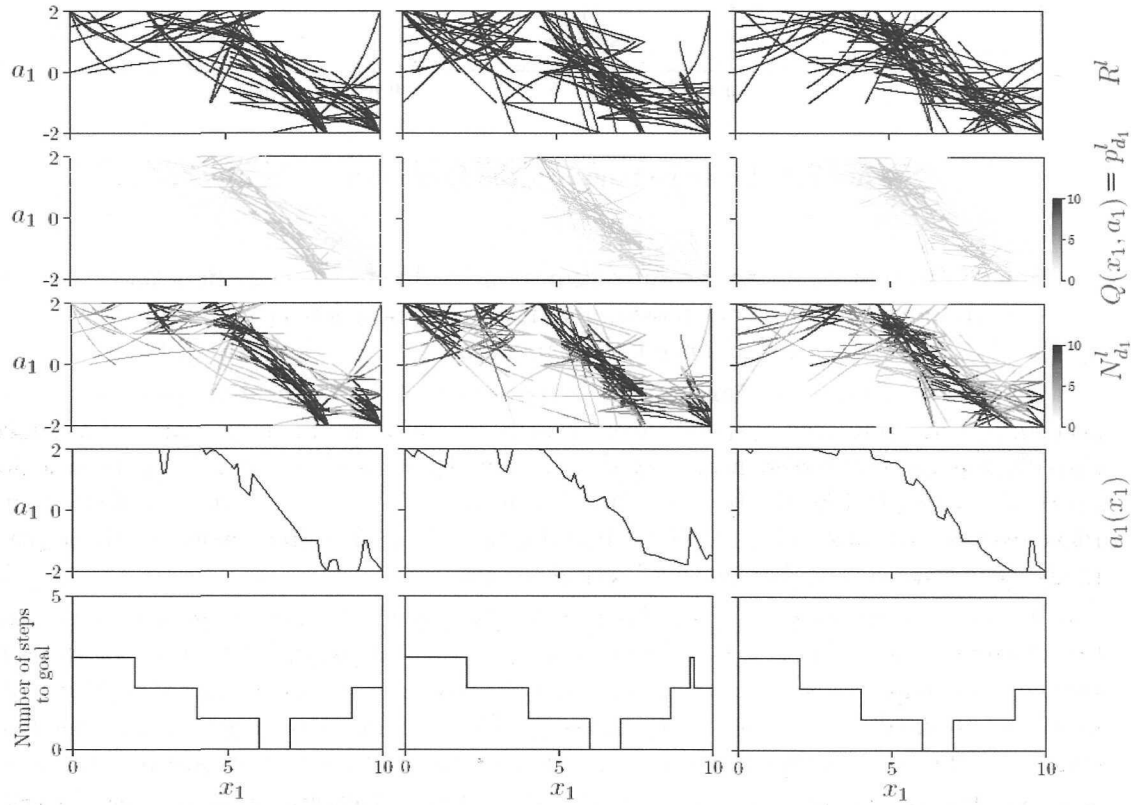


Figure 7.8: Three instances of the QFCS with unfixed fuzzy sets in World_a^{1D} .

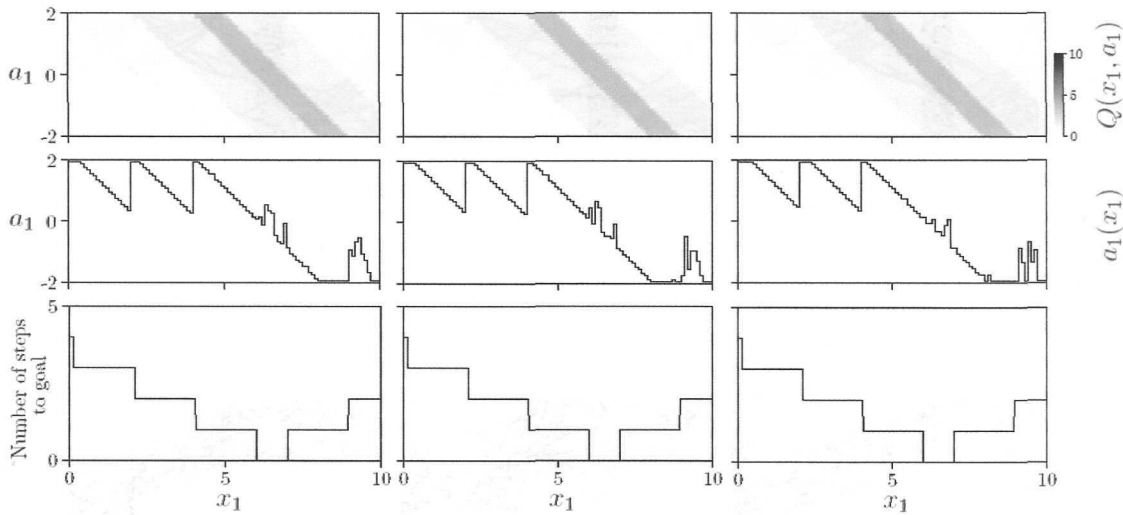


Figure 7.9: Three instances of the Q-learning in World_a^{1D} .

function is also continuous with some discontinuities due to the classifiers that represent curves with finite lengths. The steps to goal from each position x_1 is similar to the other QFCS. Therefore, the performance in this problem is similar in both QFCSs.

Figure 7.9 shows 3 instances of an experiment of 20 Q-learning systems. In this case, there are only the Q-values over a grid of 100×40 in gray scale. Therefore, the solution has many discontinuities due to that discretization but it is in the solution region according to Fig. 6.2.b. Therefore, the action function is not optimal. Comparing these results with those of the QFCSs, it can be seen how classifiers evolve to those parts in the space xa with higher reward for a given x_1 .

Figure 7.10 shows the averaged action function in the World_a^{1D} , the averaged number of steps to goal in each possible position x_1 and the averaged number of steps to goal against trials. These averages are over 20 instances of each system, QFCS with fixed and unfixed fuzzy sets and Q-learning. Fig. 7.10.a shows how the averaged solution of QFCSs is similar but differs to Q-learning. Both solutions are optimal since both of them are in the optimal solution region of the problem. QFCS learns the optimal solution that represents taking bigger steps. This is what a human would do. To Q-learning that does not matter because it reaches the goal with the minimum steps possible. Figure 7.10.b shows that the number of steps to goal is similar in both QFCSs. It also demonstrates how QFCS with the unfixed fuzzy sets makes fewer errors than QFCS with the fixed fuzzy sets. Figure 7.10.c shows that the convergence compared with Q-learning is similar. The convergence is reached at about 250000 trials. There is no better performance than Q-learning and this is because QFCS also learns using Q-learning with a discretization of $d = 5$.

This problem is more complex than the frog problem because it is multi-step and it gives reward only in the goal states. However, it was shown how QFCS was able to make classifiers follow the curvature defined by the optimal solution in spite of using a small value in $c = 4$. This is what makes QFCS robust. QFCS with unfixed fuzzy sets

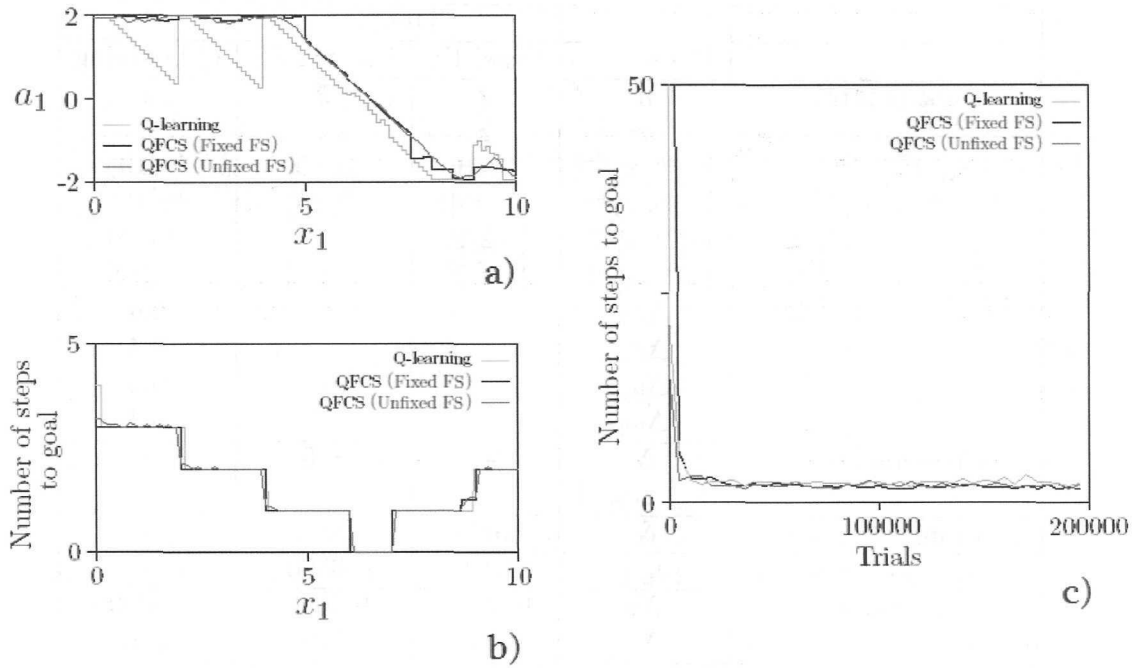


Figure 7.10: a. Average over 20 runs of the action learned in World_a^{1D}. b. Average over 20 runs of the number of steps to goal against x_1 in World_a^{1D}. c. Average over 20 runs of the number of steps to goal against trials in World_a^{1D}.

also showed to be able to follow the curvature in a better way than with the fixed fuzzy sets. This represented a good achievement in the generalization formalism of QFCS.

7.4 The 2-Environment Problem: World_a^{2D}

World_a^{2D} is an instance of the 2-Environment problem. It is a navigation task in two continuous dimensions. The objective is, starting from a position in the environment, to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous vector actions. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.4. In Q-learning, the axes x_1 and x_2 of space $x_1x_2a_1a_2$ are discretized over 25 intervals and the axes a_1 and a_2 are discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10 \times 10$ in this problem. This discretization was chosen because Q-learning does not converge with a high discretization of $100 \times 100 \times 40 \times 40$ that was used in the problems before. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to avoid that the system never reaches the goal. This number is much higher than the optimal number of steps to reach the goal. When each experiment begins, the $p_{a_1a_2}^l$ in QFCSs and the Q-values of Q-learning, are at random in $[0, 10]$. 10 is the maximum reward.

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	1	1	1
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 1]	[0, 1]	[0, 1]
	$[a_1^{\min}, a_1^{\max}]$	[0, 1]	[0, 1]	[0, 1]
	Δx_1	-	-	0.01
	Δa_1	-	-	0.01
Classifiers (Cl_i)	N	200	200	-
	$[c_{\min}, c_{\max}]$	-	[0.1, 0.4]	-
Learning Component	δ_0	500	500	-
Discovery Component	θ_{GA}	1000	3000	-
Experiments	N_T	500000	500000	500000
	N_S	1	1	1

Table 7.2: Particular parameters in the Frog Problem.

of steps N_S or less if QFCS reaches the goal before the N_S steps. If this maximal number of steps N_S is reached the trial finishes. The important variables to measure are the number of steps the system makes to reach the goal during learning that measures the convergences in learning, the action function learned by the system that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal.

7.2 The Frog Problem

The frog problem, as it was defined before, is about a frog that lives in a line. The objective is that the frog has to jump just once to catch a fly. This problem is single-step and has continuous reward because the frog is given a reward regardless of catching the fly or not. The particular parameters used in the Frog Problem for QFCS with fixed fuzzy sets, QFCS with unfixed fuzzy sets and Q-learning are shown in Table 7.2. Each axes of the space xa in Q-learning is discretized over 100 intervals. Therefore, $\Delta x = 0.01$ and $\Delta a = 0.01$. This is so, because the problem is defined in continuous variables. The size of the population is 200 which gives about 50 classifiers per activation region since there are 4. N_S is 1 since the problem is single step. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning.

At the start of each experiment, the p'_{d_1} in QFCSs and the Q-values of Q-learning are set up at random between $[0, 10]$. 10 is used because it is the maximum value obtained by reward. Thus, Figure 7.1 shows the initial states of three instances of this problem. Figure 7.1.a is an instance of QFCS with fixed fuzzy sets. Figure 7.1.b is an instance of QFCS with unfixed fuzzy sets. And Figure 7.1.c is an instance of Q-learning. In that figure, Fig. 7.1, rows represent the same concept—rules, Q-values, normalized Q-values and actions—but with different systems. Therefore, the classifiers are in the

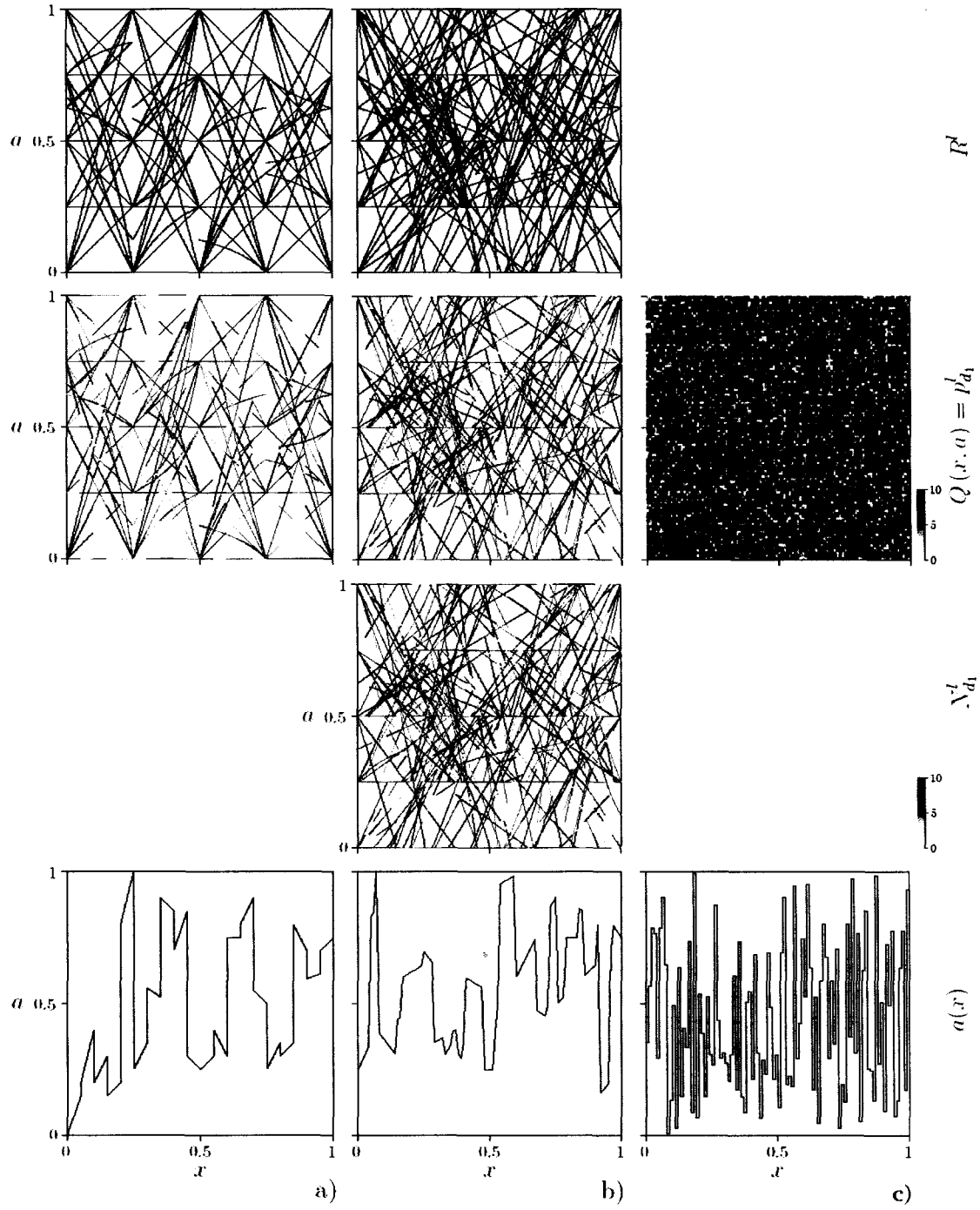


Figure 7.1: a. Initial State of the QFCS with fixed fuzzy sets in the Frog Problem. b. Initial State of the QFCS with unfixed fuzzy sets in the Frog Problem. c. Initial State of Q-learning in the Frog Problem.

first row, the prediction values or Q-values in a gray scale are in the second row, the normalized prediction values in a gray scale in the third row and the action functions in the last row. Since this problem is one-dimensional, the classifiers represent curves over space xa . These curves are obtained by drawing, for each classifier and for each possible position in the activation region of the classifier, its proposed action (the size of the jump). Thus, for each position, the classifier gives an action and these values, the position and the action, are represented using a point in the space xa . This is done for all of the positions in the activation region of the classifier, which produces a curve. This process is repeated for all of the classifiers in the population generating the curves shown in the first row. The curves represented by the classifiers are almost uniformly distributed over space xa . QFCS with the fixed fuzzy sets has the curves represented by the classifiers distributed over 4 regions that are

$$\begin{aligned}
 \text{Region}_1 &= \{(x, a) \mid (0 \leq x \leq 0.25) \wedge (0 \leq a \leq 1)\}, \\
 \text{Region}_2 &= \{(x, a) \mid (0.25 \leq x \leq 0.5) \wedge (0 \leq a \leq 1)\}, \\
 \text{Region}_3 &= \{(x, a) \mid (0.5 \leq x \leq 0.75) \wedge (0 \leq a \leq 1)\}, \\
 \text{Region}_4 &= \{(x, a) \mid (0.75 \leq x \leq 1) \wedge (0 \leq a \leq 1)\};
 \end{aligned}
 \tag{7.1}$$

while the QFCS with unfixed fuzzy set does not. This is due to the activation regions. QFCS represents the Q-function of Q-learning only in those curves determined by the classifiers. Therefore, the Q-function is not represented in all of the space xa as Q-learning demands. Moreover, it is expected that QFCS will be able to evolve classifiers to find out those regions of the space xa in the form of curves so that Q-learning is determinant for making decisions. The proposed action by any of the systems is meaningless. This occurs because the solution cannot be expected to be in the initial conditions. This problem is a bit difficult for QFCS because the goal is not a region but a point. Despite of that, QFCS provides good results.

Figure 7.2 shows 3 instances of an experiment of 20 runs of the QFCS with fixed fuzzy sets. The first row in that figure shows how classifiers evolved in such a way to represent the Q-function over the region where the Q-values are maximal for a given x . It means that the classifiers follow the optimal solution curve $a(x)$. For example, from Fig. 7.2, if $x = 0.2$ then the classifiers are arranged in such a way that they cover the action in the range $[0.4, 1]$. Classifiers do not cover all of this region but some points of it. For example point $(0.2, 0.2)$ is not covered. The curvature of the solution is controlled by the parameter c . In this problem there is no curvature and the curves represented by the classifiers do not have curvature, either. This is seen, when it is considered all the classifiers in combination as forming a thick and diffuse curve. The second row shows the prediction values $p_{a_1}^i$. This prediction values are a function of the space xa . Therefore, the values are shown in a gray scale. Black is for 10 and white for 0. These gray colors are over the curves represented by the classifiers. These learned Q-values show that the Q-function has its maximum values where the curves of the classifiers are. And the third row shows the learned actions. These action functions are not the optimal solution but QFCS does very well in spite of the discontinuities.

Figure 7.3 shows 3 instances of an experiment of 20 QFCSs with unfixed fuzzy sets. The classifiers evolved with less strength. It means that classifiers are more scattered but at the same time the classifiers over space xa seem denser. The prediction values

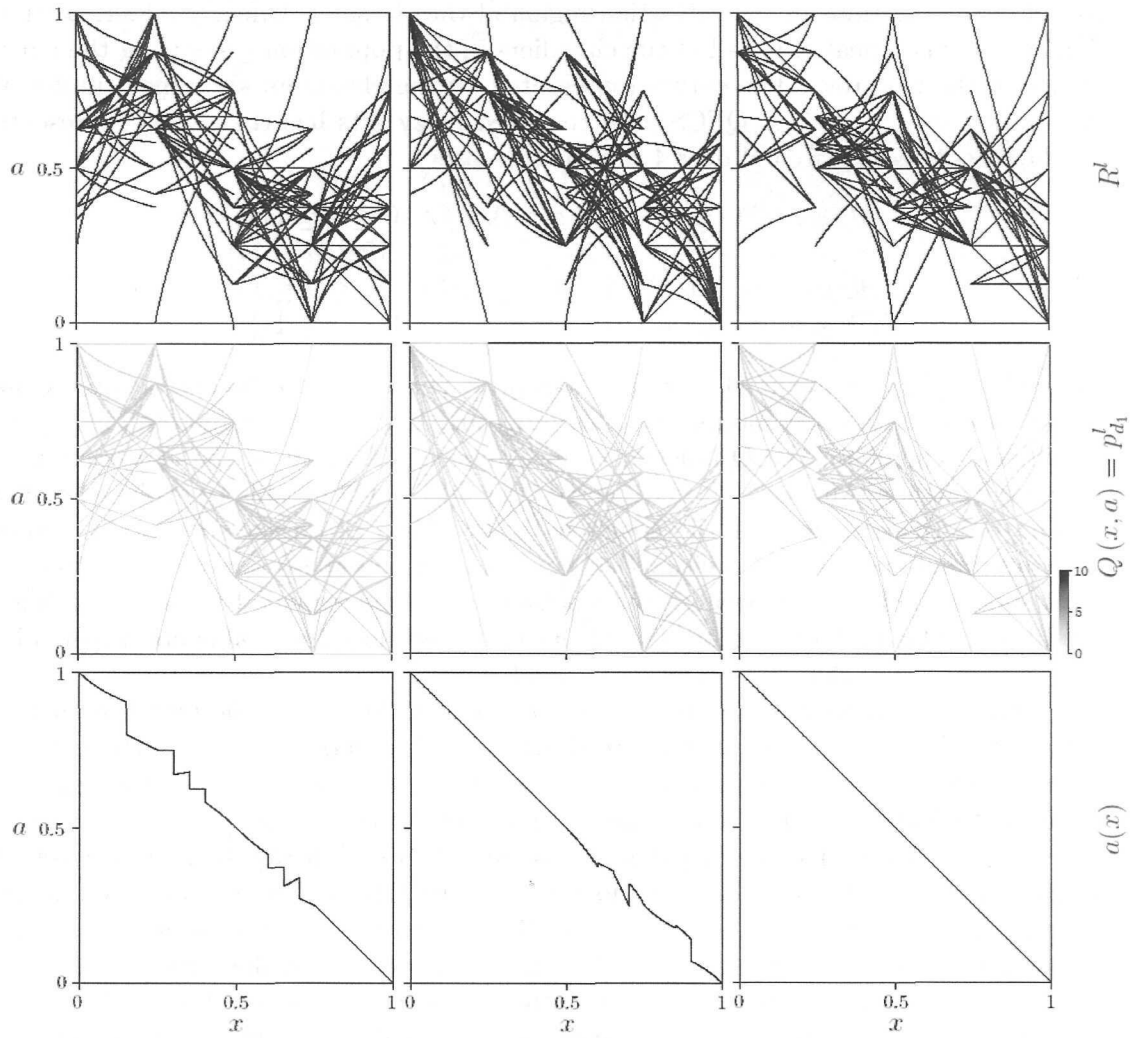


Figure 7.2: Three instances of the QFCS with fixed fuzzy sets in the Frog Problem.

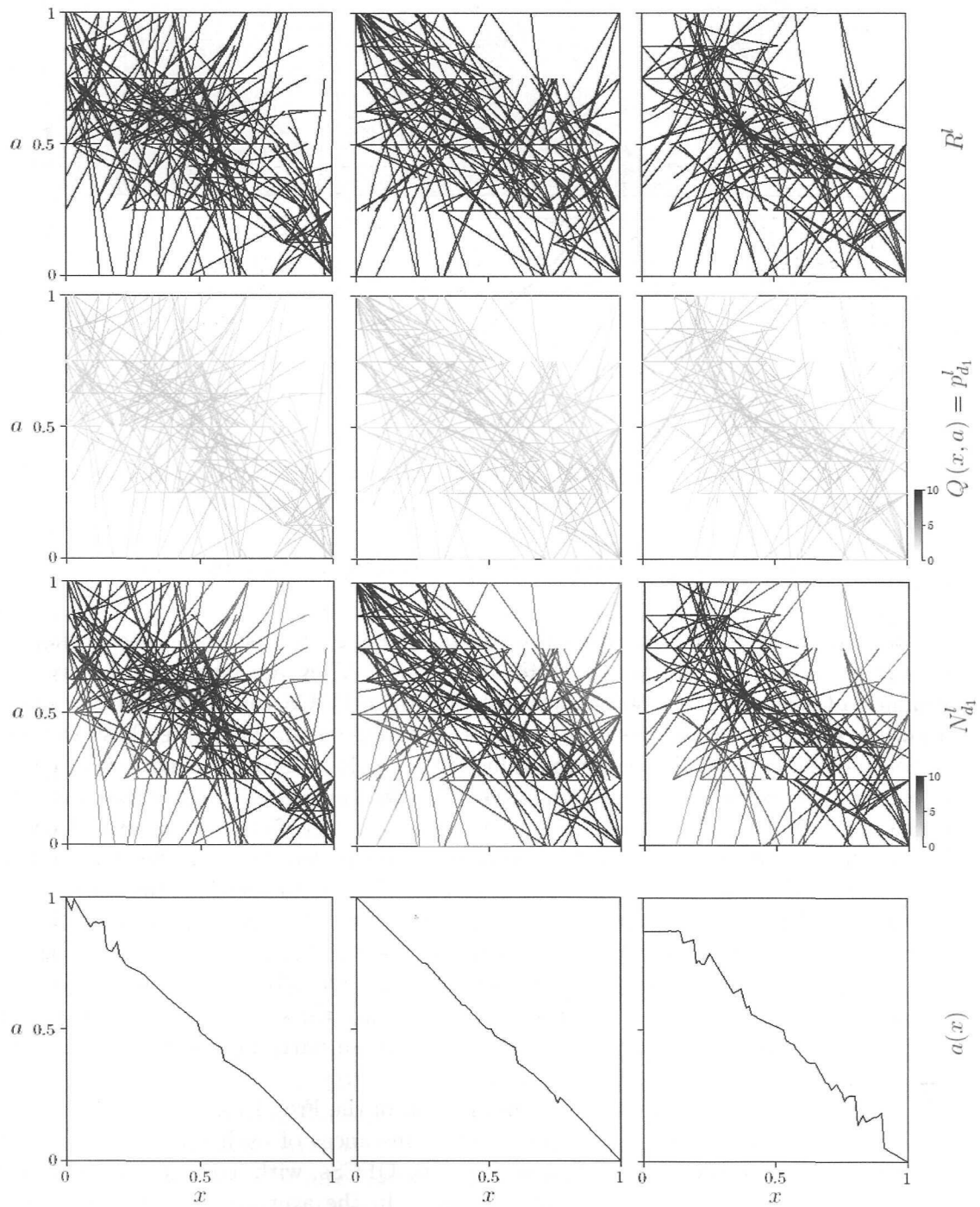


Figure 7.3: Three instances of the QFCS with unfixed fuzzy sets in the Frog Problem.

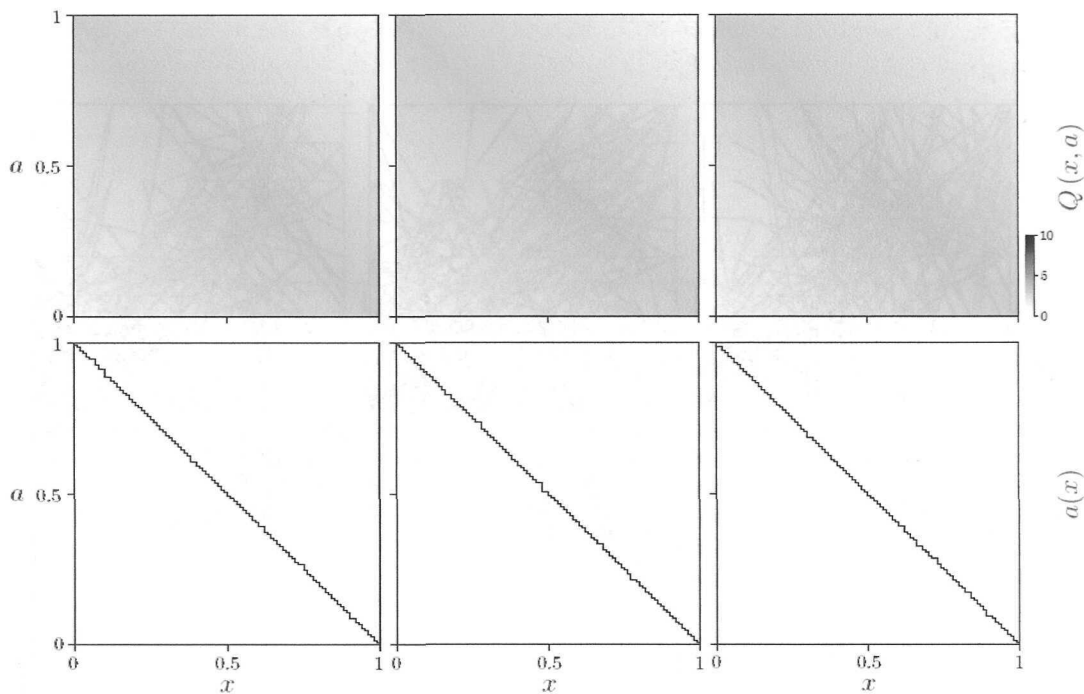


Figure 7.4: Three instances of the Q-learning in the Frog Problem.

$p_{d_1}^l$ have their maximal value over the function that is the solution. The normalized prediction values $N_{d_1}^l$ are shown in the third row. They are always in a gray scale similar to Q-values. These values were introduced in QFCS with unfixed fuzzy sets to allow the GA evolve classifiers over the solution action curve. Thus, they have to reach, per each possible state, the maximum value of reward 10. Thus, the normalized function is bigger close to the solution action curve. The discontinuities in the action functions continue to exist. These results show that the extra degree of freedom introduced by letting the activation regions evolve complicate the problem because the classifiers are more scattered compared with those obtained in QFCS with the fixed fuzzy sets.

Figure 7.4 shows 3 instances of an experiment of 20 Q-learning systems. In this case, there are only the Q-values over a grid of 100×100 in gray scale. It can be seen how the Q-values are similar to the reward function. Therefore, the solution has many discontinuities due to that discretization. Comparing these results with those of the QFCSs, it can be seen how classifiers evolve to those parts in space xa with higher reward.

Figure 7.5 shows the average action function of the Frog Problem and the average error obtained during the experiments over 20 instances of each system, QFCS with fixed and unfixed fuzzy sets and Q-learning. The QFCSs, with fixed and unfixed fuzzy sets, have similar performance in this problem. In the average error, it can be seen how QFCS is slightly better in convergence than Q-learning since QFCS decreases the error faster than Q-learning. This problem was tackled to show that QFCS is capable of solving this kind of one step problems with continuous inputs and outputs; even though QFCS was designed to deal with multistep problems.

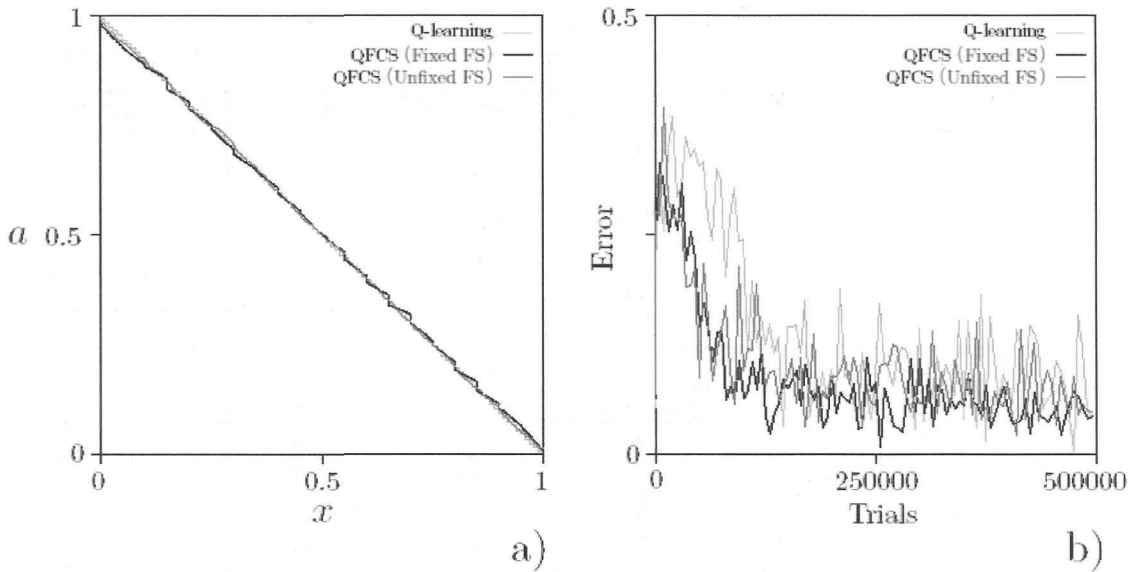


Figure 7.5: a. Average over 20 runs of the action learned in the Frog Problem. b. Average over 20 runs of the error in the Frog Problem.

These results show how QFCS can solve the frog problem and that was the objective because of the importance of that problem in LCSs literature. In comparison, QFCS and Q-learning have a similar performance. It means they converge similar and with almost the same results. But it is important to take into account that QFCS learns the action function by parts that are continuous while Q-learning learns the discretized action. On the other hand, QFCS is expected to model more complex problems where a high discretization is needed and Q-learning would not converge. The next section describes results with some instances of the n -Environment Problem.

7.3 The 1-Environment Problem: $World_a^{1D}$

$World_a^{1D}$ is an instance of the 1-Environment problem. It is a navigation task in one continuous dimension. The objective is from some position in the environment to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous actions. The particular parameters used for QFCS with fixed fuzzy sets, QFCS with unfixed fuzzy sets and Q-learning are presented in Table 7.3. The axes of space xa in Q-learning are also discretized over 100 intervals. Therefore, $\Delta x = 0.01$ and $\Delta a = 0.01$. This is so because the problem is defined over continuous variables. The size of the population is 200 which gives about 50 classifiers per activation region since there are 4. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. When each experiment begins, the $p_{d_1}^l$ in QFCSs and the Q-values of Q-learning are also set up at random between $[0, 10]$. 10 is used because it is the maximum value obtained by reward. Thus, Figure 7.6 shows the initial states of three instances of this problem. Similarly to the Frog Problem, Fig.

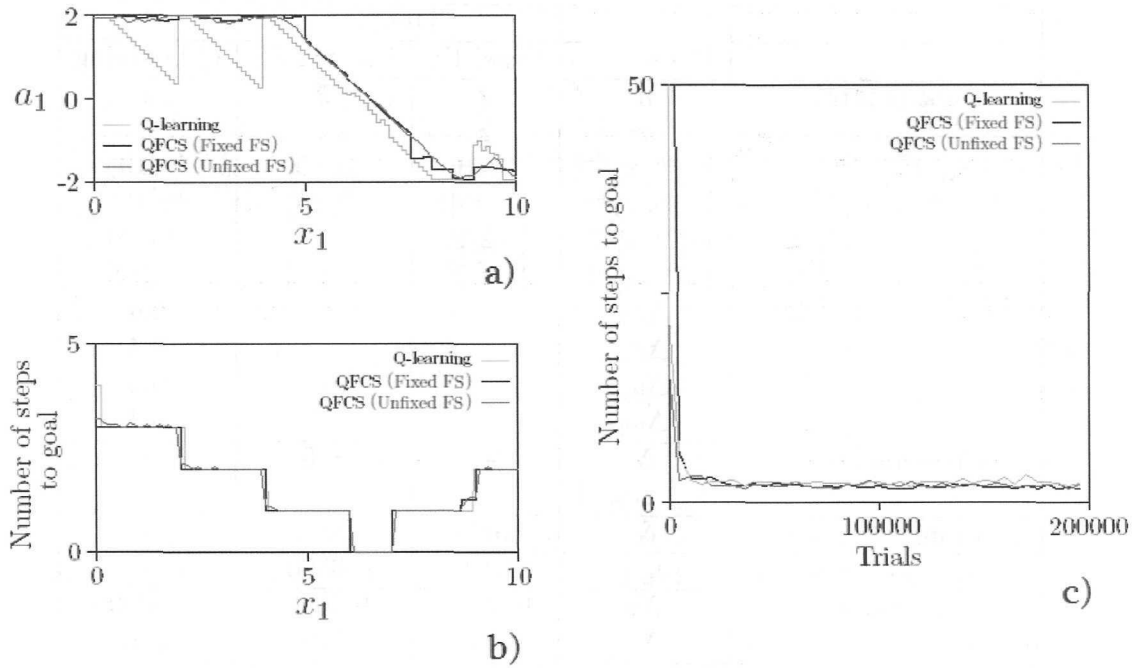


Figure 7.10: a. Average over 20 runs of the action learned in World_a^{1D}. b. Average over 20 runs of the number of steps to goal against x_1 in World_a^{1D}. c. Average over 20 runs of the number of steps to goal against trials in World_a^{1D}.

also showed to be able to follow the curvature in a better way than with the fixed fuzzy sets. This represented a good achievement in the generalization formalism of QFCS.

7.4 The 2-Environment Problem: World_a^{2D}

World_a^{2D} is an instance of the 2-Environment problem. It is a navigation task in two continuous dimensions. The objective is, starting from a position in the environment, to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous vector actions. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.4. In Q-learning, the axes x_1 and x_2 of space $x_1x_2a_1a_2$ are discretized over 25 intervals and the axes a_1 and a_2 are discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10 \times 10$ in this problem. This discretization was chosen because Q-learning does not converge with a high discretization of $100 \times 100 \times 40 \times 40$ that was used in the problems before. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to avoid that the system never reaches the goal. This number is much higher than the optimal number of steps to reach the goal. When each experiment begins, the $p_{a_1a_2}^l$ in QFCSs and the Q-values of Q-learning, are at random in $[0, 10]$. 10 is the maximum reward.

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	2	2	2
	m	2	2	2
	$[x_1^{\min}, x_1^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[x_2^{\min}, x_2^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[a_1^{\min}, a_1^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	$[a_2^{\min}, a_2^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	Δx_1	-	-	0.4
	Δx_2	-	-	0.4
	Δa_1	-	-	0.4
Δa_2	-	-	0.4	
Classifiers (Cl_i)	N	800	800	-
	$[c_{\min}, c_{\max}]$	-	[1, 4]	-
Learning Component	δ_0	1000	2000	-
Discovery Component	θ_{GA}	5000	10000	-
Experiments	N_T	1000000	1000000	1000000
	N_S	200	200	200

Table 7.4: Particular parameters in World $_a^{2D}$ and World $_b^{2D}$.

Figure 7.11 shows the vector field in the initial conditions of the QFCSs and Q-learning. These vector fields show how the classifiers really represent random vector fields at the beginning. Classifiers are not shown here because they represent vector fields with their two components functions of two variables. Q-values have the same problem because they have to be represented in four dimensions.

Figure 7.12 shows 3 instances of QFCS with fixed fuzzy sets from an experiment with 20 QFCSs. The vector fields are in the first row and the graphics of the number of steps to reach the goal from each possible state x_1x_2 are in the second row. The graphics of the number of steps to goal are two-dimensional functions represented in a gray scale. Black is for 10 steps and white for 0 steps. It can be seen how the vector fields learned by pointing to the goal. This shows that QFCS is not only looking for reaching the goal with the minor number of steps but also reducing the distance to goal. Thus the vector field is smooth. This is also shown in the second row where the number of steps reflects this behavior generating rings contrary to what happen in Fig. 6.3.b where that function formed square rings. The maximum number of steps to goal is 5. This is not the optimal solution, but it is a good solution. Thus, QFCS with fixed fuzzy sets is able to learn a vector field to reach the goal. This vector field as it has been explained comes from a set of continuous vectors. Therefore, the learned vector field is continuous. It is obvious from the structure of classifiers that there are going to be discontinuities, but as it was said before, the solution of this problem is not a vector field but a region of vector fields similar to that in World $_a^{1D}$. It means that in each state x_1x_2 there are many vector solutions. Parameter c now has to measure the curvature of a vector field. The vector field introduces two functions of the state spaces x_1x_2 that

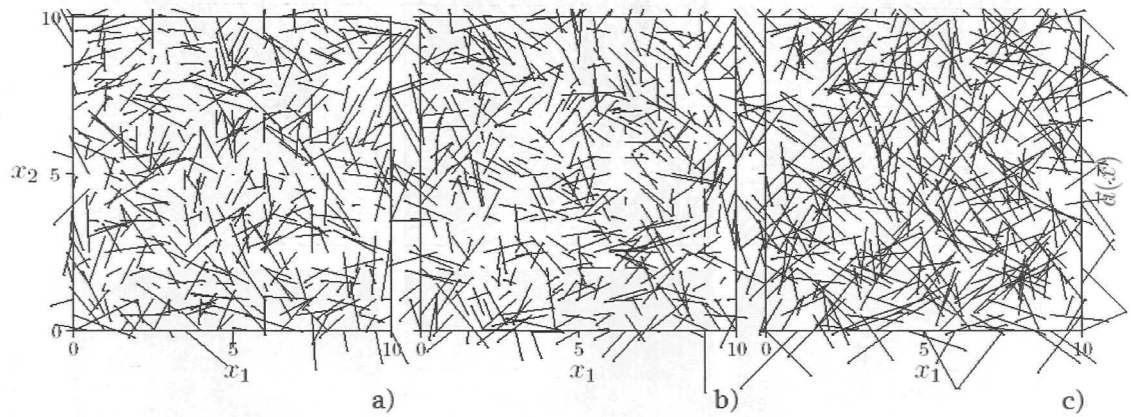


Figure 7.11: a. Initial State of the QFCS with fixed fuzzy sets in $WORLD_a^{2D}$. b. Initial State of the QFCS with unfixed fuzzy sets in $WORLD_a^{2D}$. c. Initial State of Q-learning in $WORLD_a^{2D}$.

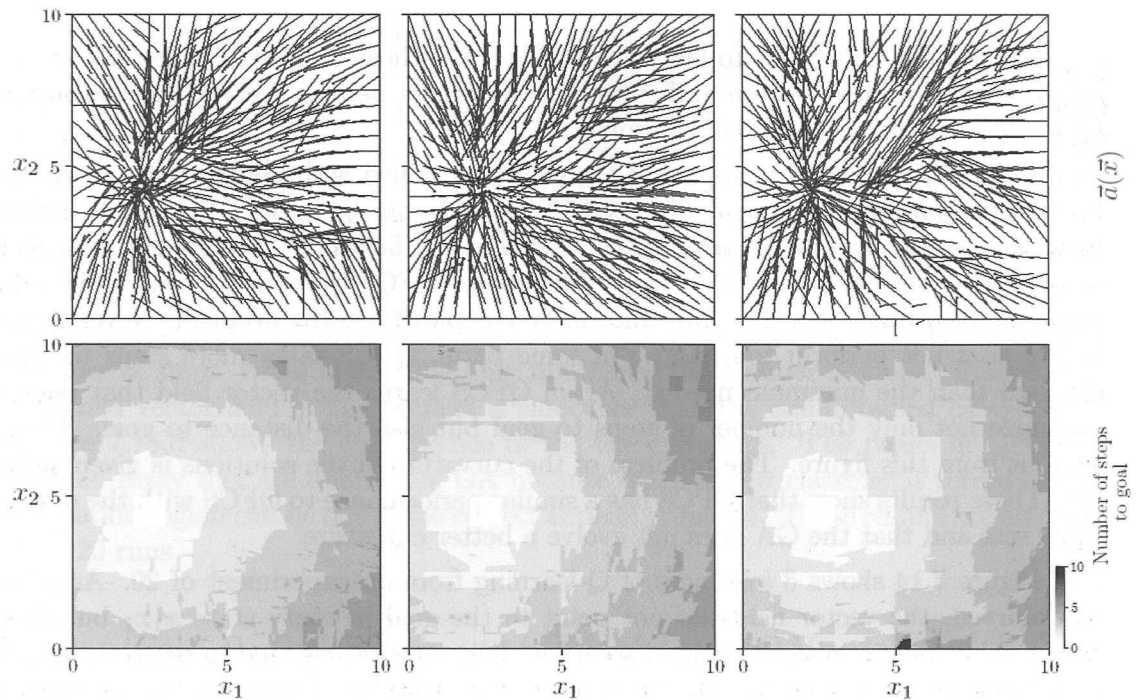


Figure 7.12: Three instances of the QFCS with fixed fuzzy sets in $WORLD_a^{2D}$.

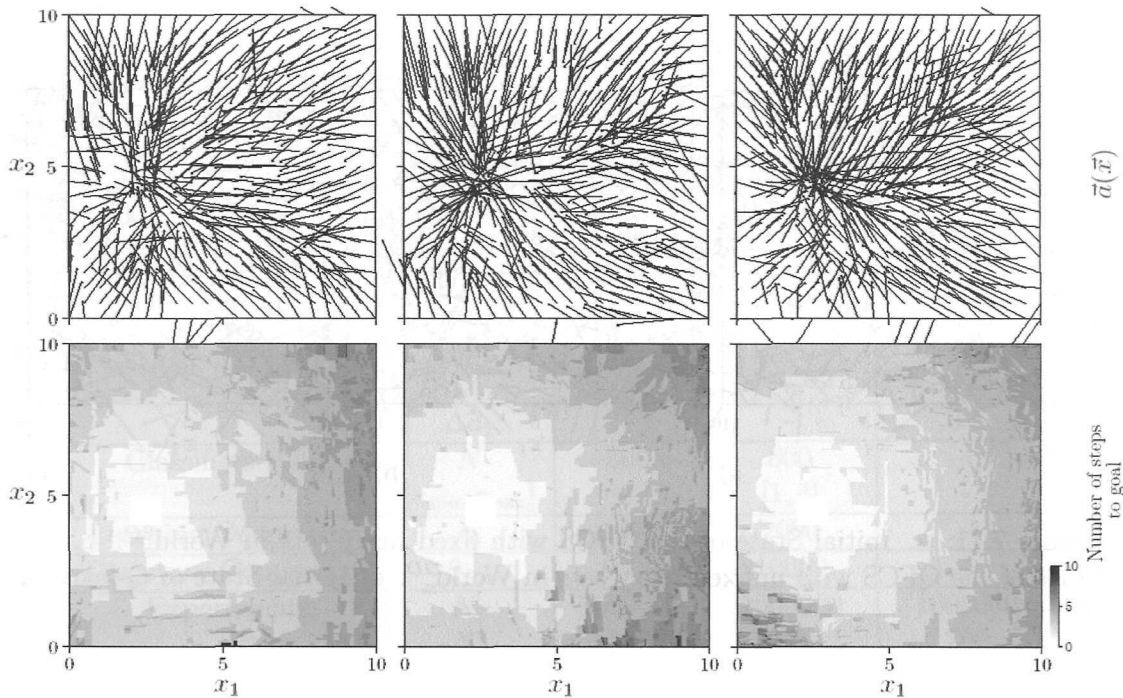


Figure 7.13: Three instances of the QFCS with unfixed fuzzy sets in World_a^{2D} .

are surfaces. So, QFCS has to learn the curvature of those surfaces. Since they are two surfaces, the problem is more difficult than the others before. This is the reason why QFCS learns only sub-optimal solutions.

Figure 7.13 shows 3 instances of the results obtained with QFCS with the unfixed fuzzy sets from an experiment of 20 QFCSs. As it can be seen, QFCS with unfixed fuzzy sets is also capable of learning the needed vector field but it has some problems in certain small regions. Thus, it is more difficult for QFCS to learn the vector field due to the unfixed fuzzy sets that introduce more complexity to the problem. Nevertheless, QFCS does not perform too badly. In some parts, QFCS takes more steps to reach the goal than the minimum needed. Again QFCS learns the vector field that tries to minimize not only the number of steps to goal but also the distance to goal. This is obvious from this figure. The problem of the curvature of the solutions is given to the GA. These results show that QFCS has a similar performance to QFCS with the unfixed fuzzy sets and that the GA does not evolve a better curvature.

Figure 7.14 shows 3 instances of Q-learning from an experiment of 20. As it can be observed, the vector fields do not point to the goal in every state x_1x_2 but those vector fields give an almost optimal solution. It is not optimal due to discretization of Q-function and that that discretization is not high enough to represent the problem in its continuous nature. That is one of the reasons for the need of new algorithms that can deal with continuous problems. At this point Q-learning does better than QFCS, but this is a first step in this direction. It is remarkable that QFCS also minimized the distance while Q-learning does not. This is important because that is part of the

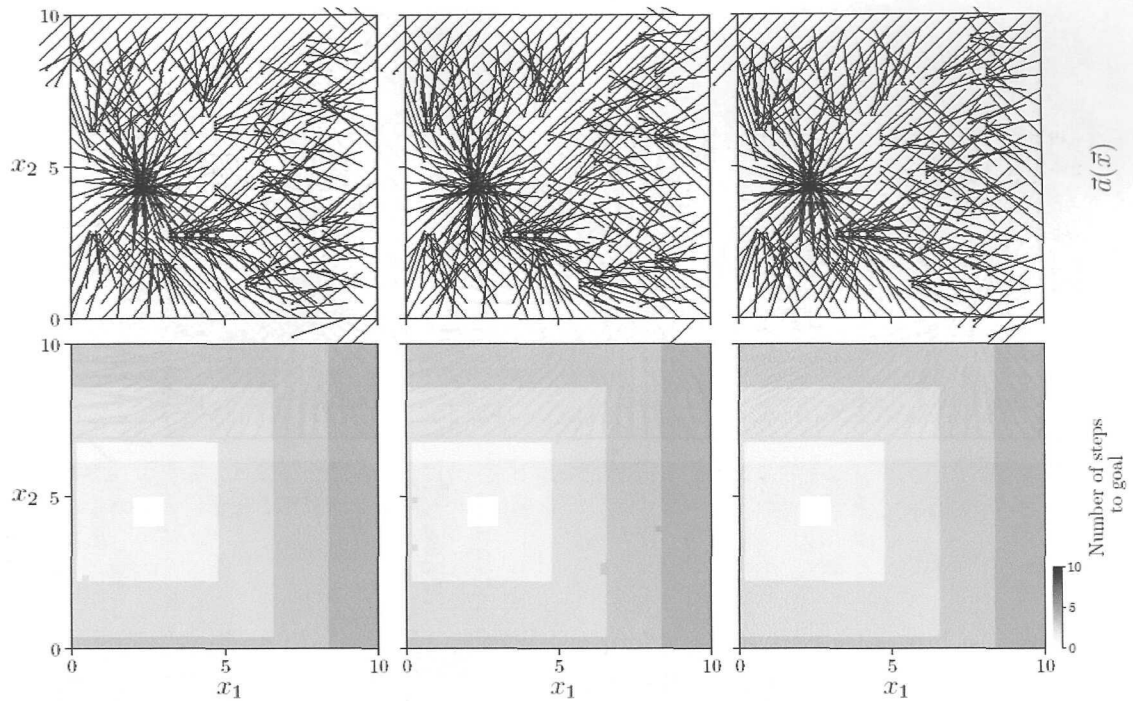


Figure 7.14: Three instances of Q-learning in $World_a^{2D}$.

intelligent behavior. A dog does not get for the food following whichever route with less steps but the one with the minimum distance. If both minimize the number of steps, and the distance is taken into account, the system would seem more intelligent. QFCS does not have anything in its formalism that minimized distance, so, it seems more intelligent. This shows that QFCS is giving good new lines to follow.

Figure 7.15 shows the average of the obtained results by the QFCSs and the Q-learning system. Figures 7.15.a, 7.15.b and 7.15.c show the averages obtained by the QFCS with the fixed fuzzy sets, QFCS with the unfixed fuzzy sets and Q-learning respectively. These vector fields demonstrate how QFCS minimized the distance, not only the number of steps, while Q-learning only minimizes the number of steps. Therefore, the obtained vector field in the average is smoother than the one obtained by Q-learning. Figure 7.15.d shows the convergence of the three systems. The peaks in QFCS with unfixed fuzzy sets are due to the GA because when a new classifier is introduced, the activation region can be out of the correct actions proposed. These results are an average over 20 runs.

7.5 The 2-Environment Problem: $World_b^{2D}$

$World_b^{2D}$ is an instance of the 2-Environment problem. It is also a navigation task in two continuous dimensions. The objective is, starting from a position in the environment, to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous vector actions. The difference with $World_a^{2D}$

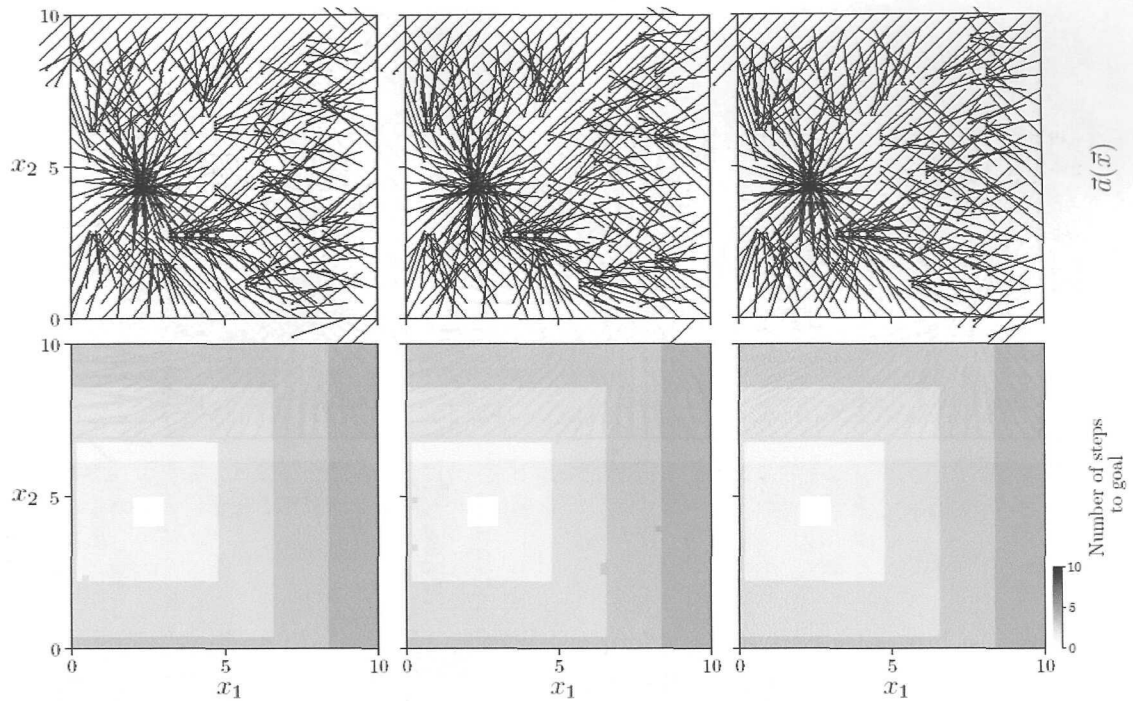


Figure 7.14: Three instances of Q-learning in $World_a^{2D}$.

intelligent behavior. A dog does not get for the food following whichever route with less steps but the one with the minimum distance. If both minimize the number of steps, and the distance is taken into account, the system would seem more intelligent. QFCS does not have anything in its formalism that minimized distance, so, it seems more intelligent. This shows that QFCS is giving good new lines to follow.

Figure 7.15 shows the average of the obtained results by the QFCSs and the Q-learning system. Figures 7.15.a, 7.15.b and 7.15.c show the averages obtained by the QFCS with the fixed fuzzy sets, QFCS with the unfixed fuzzy sets and Q-learning respectively. These vector fields demonstrate how QFCS minimized the distance, not only the number of steps, while Q-learning only minimizes the number of steps. Therefore, the obtained vector field in the average is smoother than the one obtained by Q-learning. Figure 7.15.d shows the convergence of the three systems. The peaks in QFCS with unfixed fuzzy sets are due to the GA because when a new classifier is introduced, the activation region can be out of the correct actions proposed. These results are an average over 20 runs.

7.5 The 2-Environment Problem: $World_b^{2D}$

$World_b^{2D}$ is an instance of the 2-Environment problem. It is also a navigation task in two continuous dimensions. The objective is, starting from a position in the environment, to reach another one called goal where a reward is given. This problem is important because is multi-step and with a set of continuous vector actions. The difference with $World_a^{2D}$

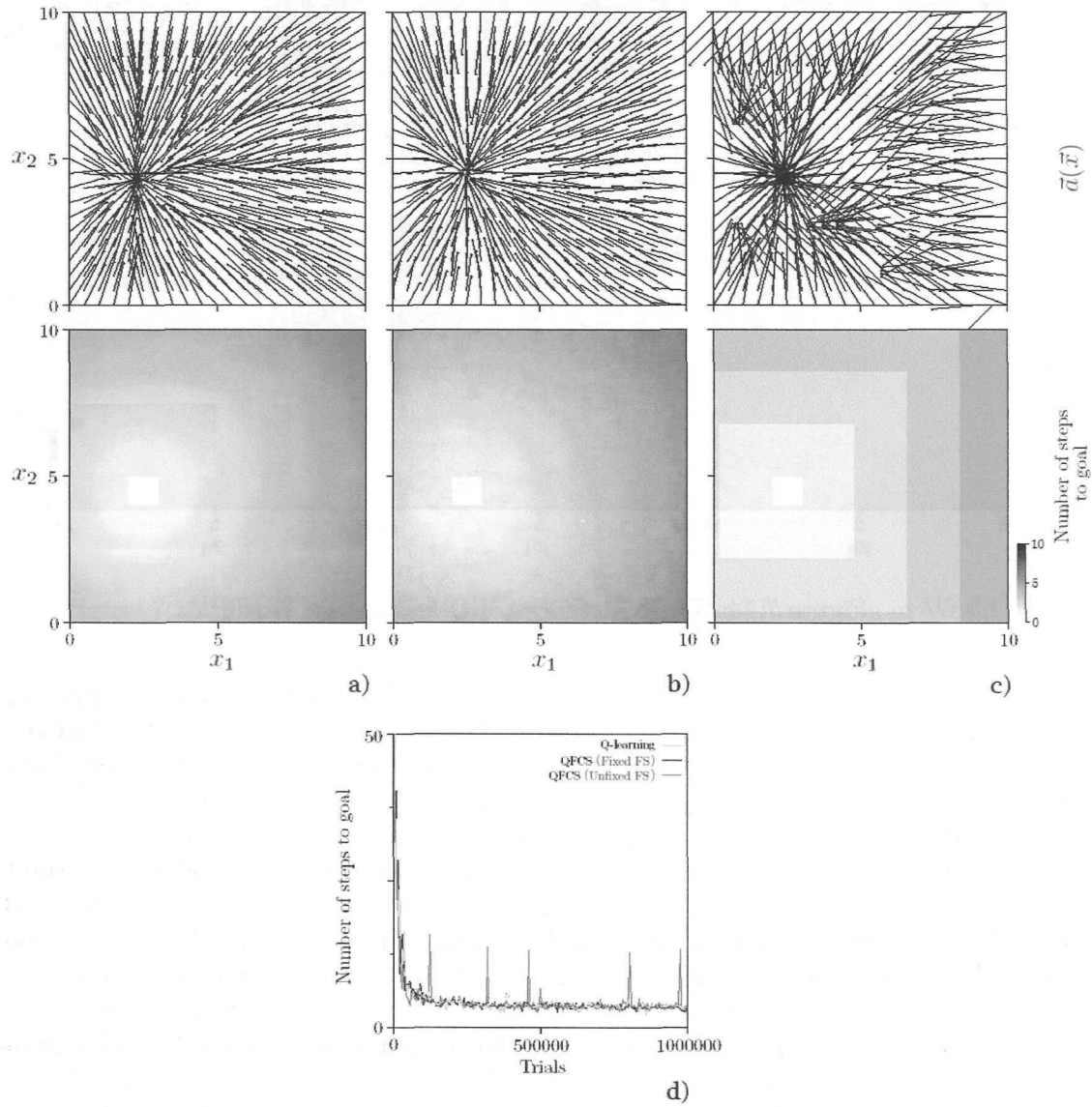


Figure 7.15: a. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with QFCS with fixed fuzzy sets. b. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with QFCS with unfixed fuzzy sets. c. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained with Q-learning. d. Average over 20 runs of the number of steps to goal against trials in World^{1D}.

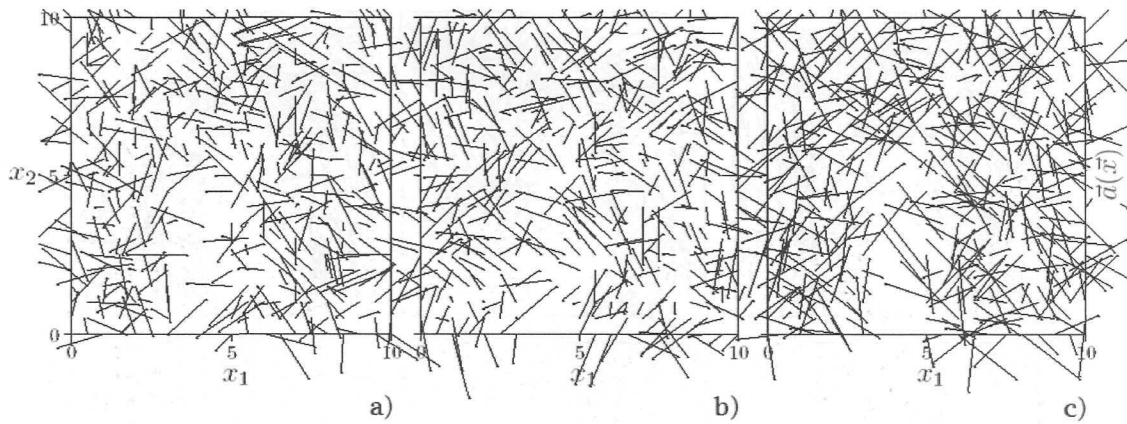


Figure 7.16: a. Initial State of the QFCS with fixed fuzzy sets in $World_b^{2D}$. b. Initial State of the QFCS with unfixed fuzzy sets in $World_b^{2D}$. c. Initial State of Q-learning in $World_b^{2D}$.

is the addition of an obstacle that makes the problem more complex and realistic. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are also presented in Table 7.4. These parameters are the same as those in $World_a^{2D}$ problem because the only difference between $World_b^{2D}$ and $World_a^{2D}$ is the introduction of an obstacle and the change of the goal.

Figure 7.16 shows the vector fields in the initial conditions of the QFCSs and Q-learning. These vector fields also show how the classifiers really represent random vector fields at the beginning. In the obstacle position, the vector field is not drawn because that is a region the systems cannot access. Therefore, the vector field there does not matter. The obtained vector field will depend on its neighborhood vector field.

Figure 7.17 shows 3 instances of QFCS with fixed fuzzy sets from an experiment with 20 QFCSs. The vector fields are shown in the first row and the graphics of the number of steps to reach the goal from each possible state x_1x_2 are in the second row. The graphics of the number of steps to goal are two-dimensional functions represented in a gray scale. Black is for 10 steps and white for 0 steps. It can be seen how the vector fields learned by pointing in the road to goal. This shows that QFCS is not only looking for reaching the goal with the minor number of steps but also reducing the distance to goal. Thus the vector field is smooth. This is also shown in the second row where the number of steps reflects this behavior generating rings around the goal contrary to what happen in Fig. 6.4.b where that function formed square rings around the goal. The maximum number of steps to goal is 10. This is not the optimal solution, but it is a good solution. Thus, QFCS with fixed fuzzy sets is able to learn a vector field to reach the goal. This vector field, as it has been explained, comes from a set of continuous vectors. Therefore, the learned vector field is also continuous. It is also obvious, from the structure of classifiers, that there are going to be discontinuities but as it was said before the solution of this problem is not a vector field but a region of vector fields similar to that in $World_a^{1D}$. It means that in each state x_1x_2 there are many vectors as solutions. Parameter c now has to measure the curvature of a vector field. The vector

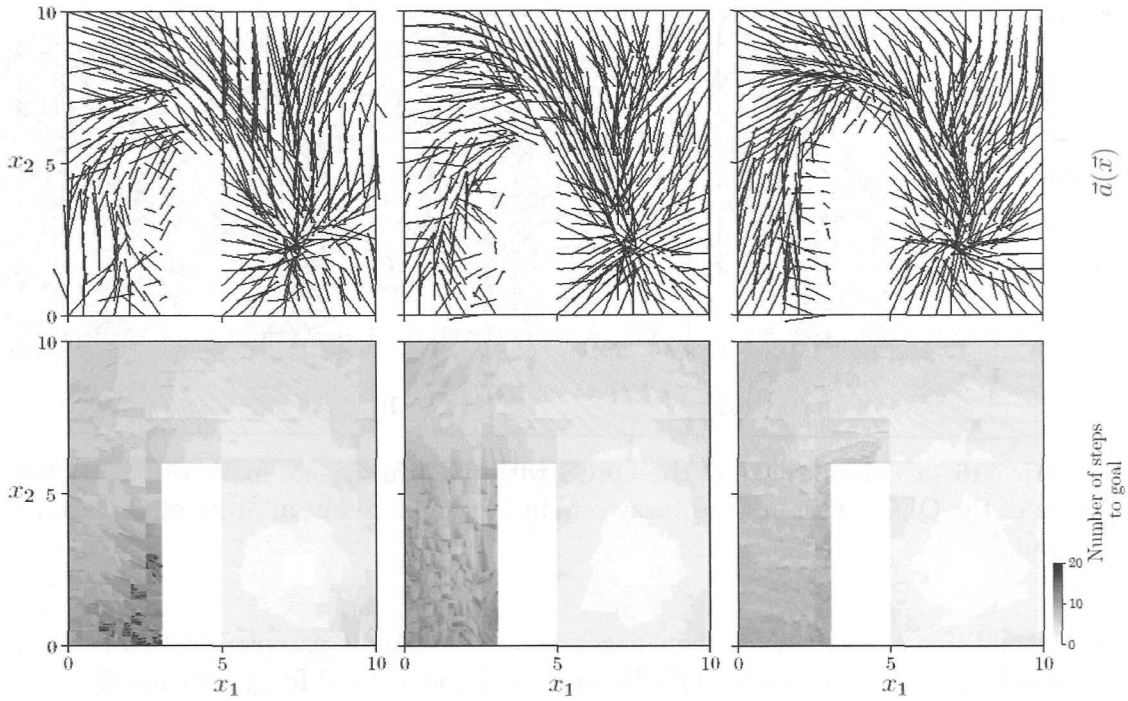


Figure 7.17: Three instances of the QFCS with fixed fuzzy sets in World_b^{2D} .

field introduces two functions of the state spaces x_1x_2 that are surfaces, so, QFCS has to learn the curvature of those surfaces. Since they are two surfaces, the problem is more difficult than the others before. This is a reason why QFCS learns only suboptimal solutions. Parameter c appears to work well.

Figure 7.18 shows 3 instances of the obtained results with QFCS with the unfixed fuzzy sets from an experiment of 20 QFCSs. As it can be seen, QFCS with unfixed fuzzy sets is also capable of learning the vector field needed but it has some problems in regions far from the goal. Thus, it is more difficult for QFCS to learn the vector field due to the unfixed fuzzy sets that introduce more complexity to the problem. Nevertheless, QFCS does not perform too badly. In some parts, QFCS takes more steps to reach the goal than the minimum needed. Again QFCS learns the vector field that tries to minimize not only the number of steps to goal but also the distance to goal. This is obvious from this figure. The problem of the curvature of the solutions is given to the GA. These results show that QFCS has a similar performance to QFCS with the unfixed fuzzy sets and that the GA does not evolve a better curvature.

Figure 7.19 shows 3 instances of Q-learning from an experiment of 20. As it can be observed, the vector fields do not point through the road to the goal in every state x_1x_2 but those vector fields give almost optimal solutions. It is not optimal due to the discretization of Q-function and that that discretization is not high enough to represent the problem in its continuous nature.

Figure 7.20 shows the average of the obtained results by the QFCSs and the Q-learning system. Figures 7.20.a, 7.20.b and 7.20.c show the averages obtained by the

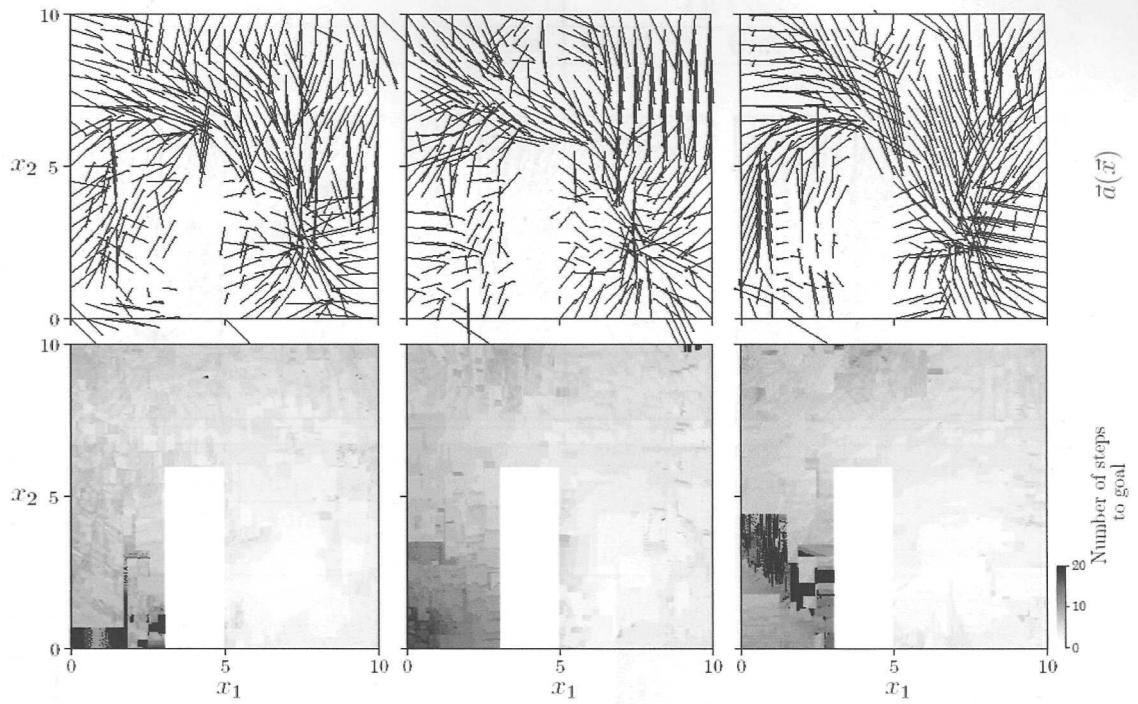


Figure 7.18: Three instances of the QFCS with unfixed fuzzy sets in World_b^{2D}.

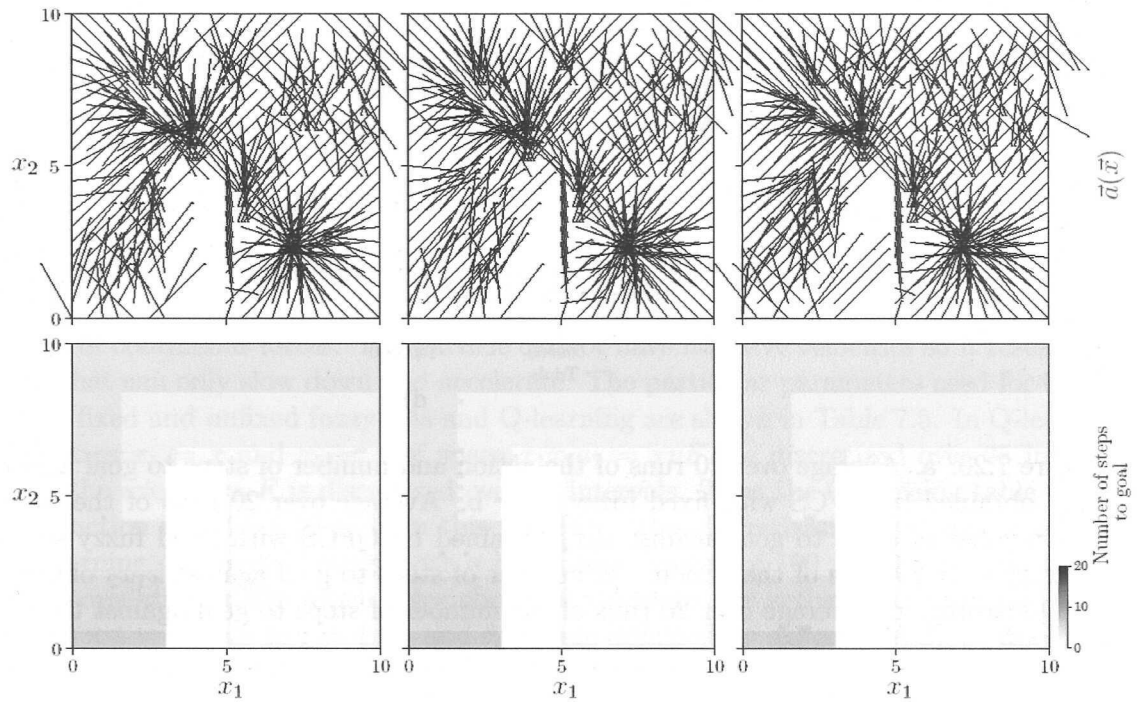


Figure 7.19: Three instances of Q-learning in World_b^{2D}.

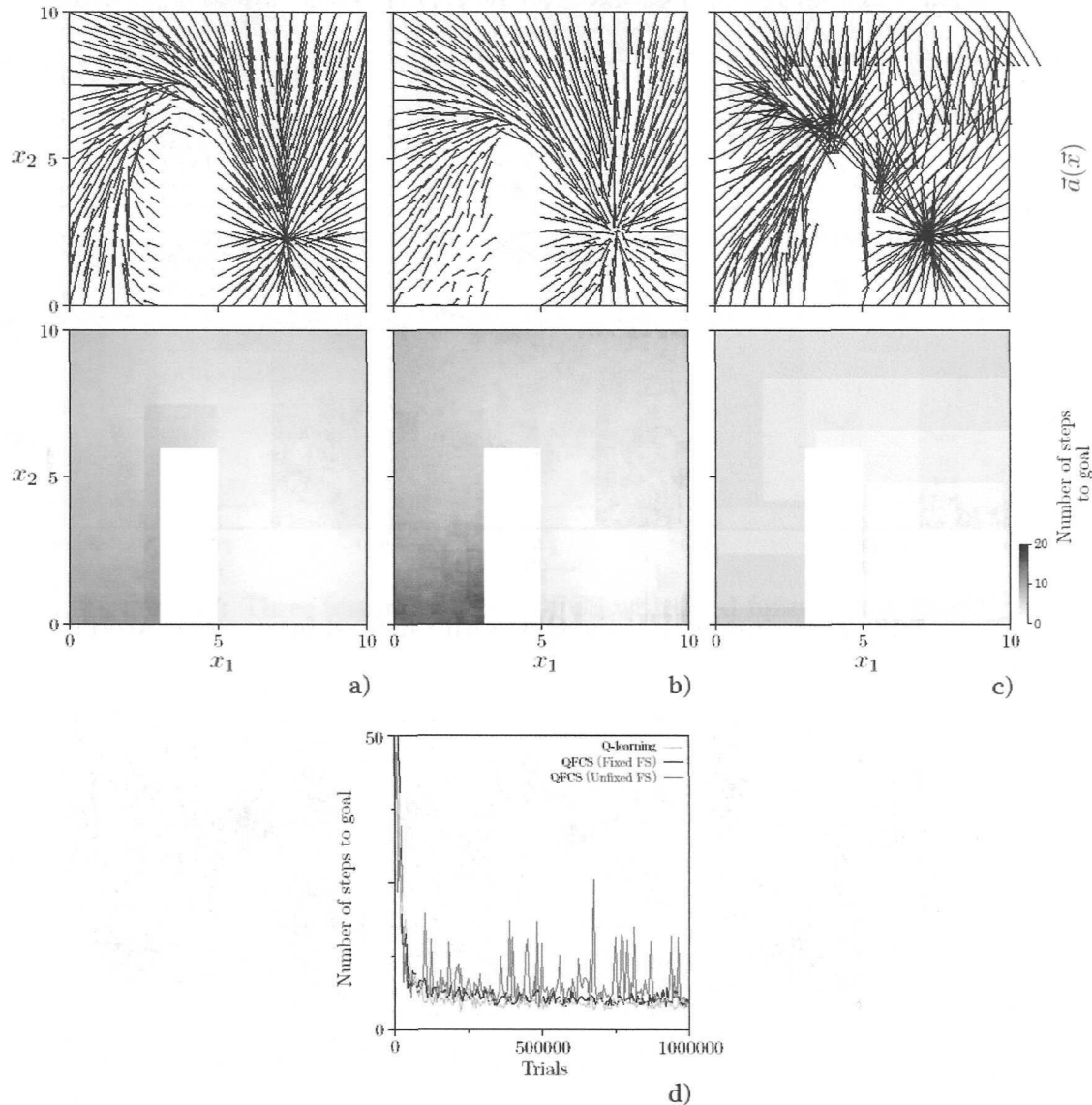


Figure 7.20: a. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by QFCS with fixed fuzzy sets c. Average over 20 runs of the action and number of steps to goal against x_1x_2 obtained by Q-learning. d. Average over 20 runs of the number of steps to goal against trials in World_6^{1D} .

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	2	2	2
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[x_2^{\min}, x_2^{\max}]$	[0, 5]	[0, 5]	[0, 5]
	$[a_1^{\min}, a_1^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	Δx_1	-	-	0.4
	Δx_2	-	-	0.2
	Δa_1	-	-	0.4
Classifiers (Cl_i)	N	800	800	-
	$[c_{\min}, c_{\max}]$	-	[1, 4]	-
Learning Component	δ_0	1000	2000	-
Discovery Component	θ_{GA}	5000	10000	-
Experiments	N_T	1000000	1000000	1000000
	N_S	200	200	200

Table 7.5: Particular parameters in Particle_a^{1D}.

QFCS with the fixed fuzzy sets, QFCS with the unfixed fuzzy sets and Q-learning respectively. These vector fields demonstrate how QFCS minimized the distance not only the number of steps while Q-learning only minimizes the number of steps. Therefore, the obtained vector field in the average is smoother than the one obtained by Q-learning. Figure 7.20.d shows the convergence of the three systems. The convergence of QFCS with unfixed fuzzy sets is noisier. These results are an average over 20 runs.

7.6 The 2-Environment Problem: Particle_a^{1D}

Particle_a^{1D} is a 2-Environment problem about the simulation of an inertial particle in one dimension. The goal is to take the particle from some position to another one through a set of continuous forces. The particle cannot have negative velocities so it resembles a car that can only slow down and accelerate. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.5. In Q-learning, the axes $x_1 = x$ and $x_2 = v$ of space $x_1x_2a_1 = xvF$ are discretized over 25 intervals and the axes $a_1 = F$ is discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10$ in this problem. This discretization was chosen because Q-learning does not converge with a high discretization of $100 \times 100 \times 40$. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to avoid the system to never reach the goal. This number is much higher than optimal number of steps to reach the goal. When each experiment begins, the $p_{d_1d_2}^l$ in QFCSs and the Q-values of Q-learning are also set up at random in $[0, 10]$. 10 is the maximum reward possible.

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	2	2	2
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[x_2^{\min}, x_2^{\max}]$	[0, 5]	[0, 5]	[0, 5]
	$[a_1^{\min}, a_1^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	Δx_1	-	-	0.4
	Δx_2	-	-	0.2
	Δa_1	-	-	0.4
Classifiers (Cl_i)	N	800	800	-
	$[c_{\min}, c_{\max}]$	-	[1, 4]	-
Learning Component	δ_0	1000	2000	-
Discovery Component	θ_{GA}	5000	10000	-
Experiments	N_T	1000000	1000000	1000000
	N_S	200	200	200

Table 7.5: Particular parameters in Particle_a^{1D}.

QFCS with the fixed fuzzy sets, QFCS with the unfixed fuzzy sets and Q-learning respectively. These vector fields demonstrate how QFCS minimized the distance not only the number of steps while Q-learning only minimizes the number of steps. Therefore, the obtained vector field in the average is smoother than the one obtained by Q-learning. Figure 7.20.d shows the convergence of the three systems. The convergence of QFCS with unfixed fuzzy sets is noisier. These results are an average over 20 runs.

7.6 The 2-Environment Problem: Particle_a^{1D}

Particle_a^{1D} is a 2-Environment problem about the simulation of an inertial particle in one dimension. The goal is to take the particle from some position to another one through a set of continuous forces. The particle cannot have negative velocities so it resembles a car that can only slow down and accelerate. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.5. In Q-learning, the axes $x_1 = x$ and $x_2 = v$ of space $x_1x_2a_1 = xvF$ are discretized over 25 intervals and the axes $a_1 = F$ is discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10$ in this problem. This discretization was chosen because Q-learning does not converge with a high discretization of $100 \times 100 \times 40$. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to avoid the system to never reach the goal. This number is much higher than optimal number of steps to reach the goal. When each experiment begins, the $p_{d_1d_2}^l$ in QFCSs and the Q-values of Q-learning are also set up at random in $[0, 10]$. 10 is the maximum reward possible.

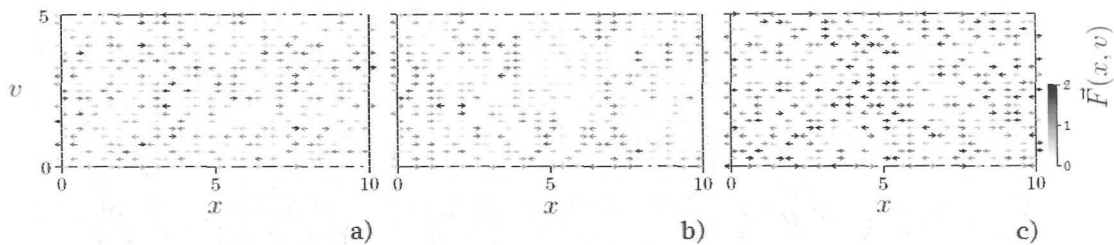


Figure 7.21: a. Initial State of the QFCS with fixed fuzzy sets in Particle $_a^{1D}$. b. Initial State of the QFCS with unfixed fuzzy sets in Particle $_a^{1D}$. c. Initial State of Q-learning in Particle $_a^{1D}$.

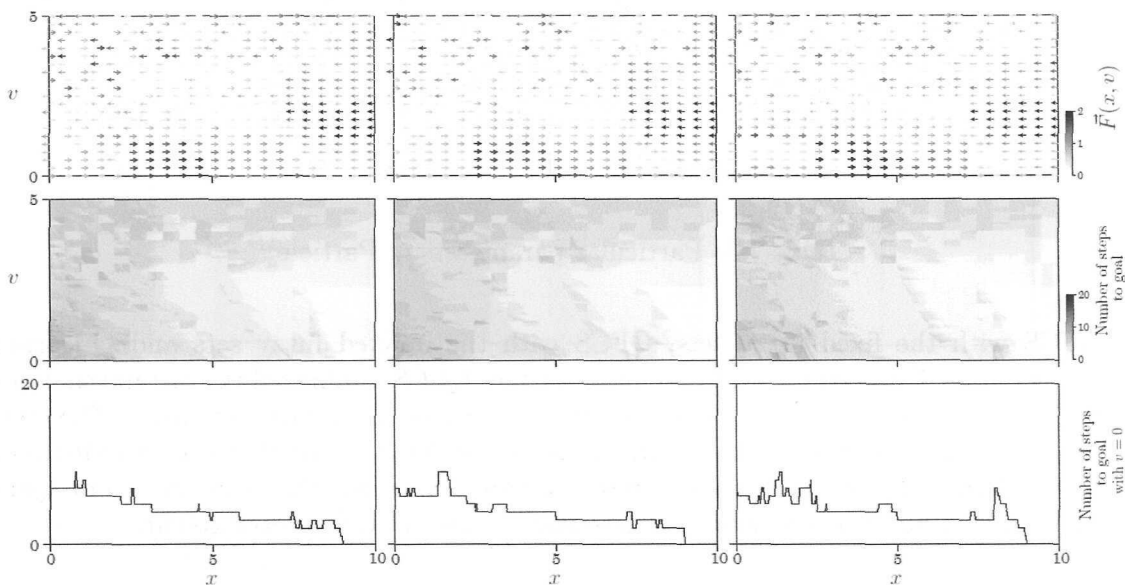


Figure 7.22: Three instances of the QFCS with fixed fuzzy sets in Particle $_a^{1D}$.

Figure 7.21 shows the vector fields in the initial conditions of the QFCSs and Q-learning. These vector fields are of one dimension and represent the force. The vector fields are represented as arrows that determine the directions of the vectors and with colors in a gray scale to show their intensities. Thus, force is applied over the particle to move it to right or left. These vector fields show how the classifiers really represent random vector fields at the beginning.

Figure 7.22 shows 3 instances of QFCS with fixed fuzzy sets from an experiment with 20 QFCSs. The vector fields are in the first row. The graphics of the number of steps to reach the goal from each possible state xv are in the second row, represented in a gray scale since they are two dimensional functions. The plots of the number of steps to reach the goal from each possible state xv with $v = 0$ are in the third row. A number of 20 in the number of steps to goal means either the system takes 20 steps or more to reach the goal, or the system simply could not reach the goal at anytime. It can be seen how QFCS learns the vector field that takes it to the goal. The learned vector field is

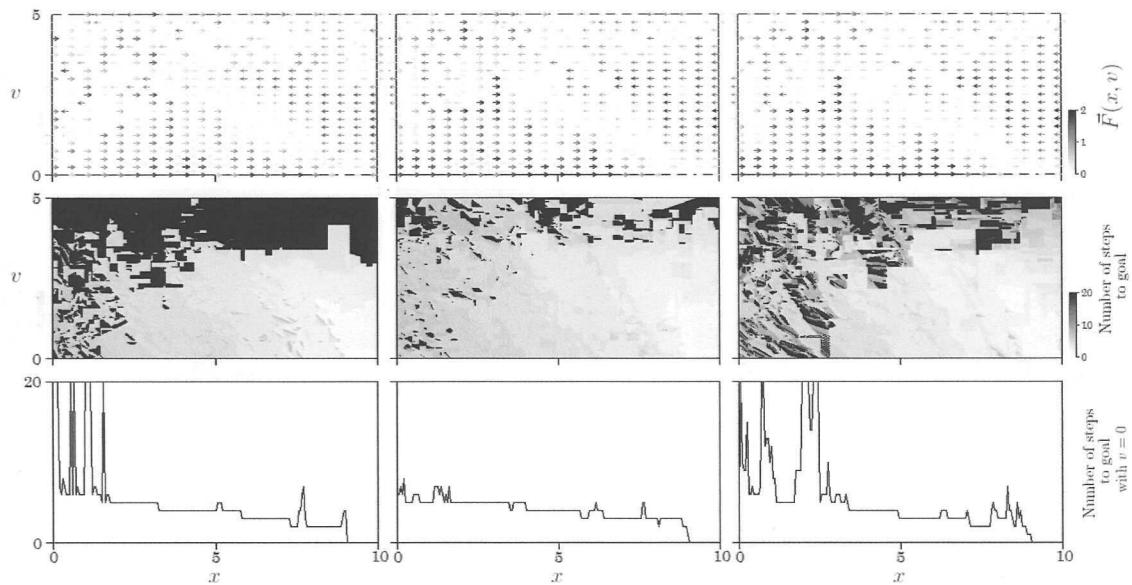


Figure 7.23: Three instances of the QFCS with unfixed fuzzy sets in Particle_a^{1D}.

noisier, farther away from the goal and the line with $v = 0$. In the graph of number of steps to goal against xv , it is shown how QFCS is capable of learning to reach the goal from another initial state xv with $v \neq 0$. This problem is more complex than the others because the QFCS is not able to move in any arbitrary direction over the state space xv since the particle has inertia. QFCS learns well how to reach the goal.

Figure 7.23 shows 3 instances of the obtained results with QFCS with the unfixed fuzzy sets from an experiment of 20 QFCSs. As it is shown, QFCS with unfixed fuzzy sets is also capable of learning the needed vector field but it has some problems in regions far from goal. This is shown in the black regions of the plots of the second row. Those regions mean that QFCS does not reach the goal in less than 20 steps if the particle initiates from a point over those black regions. Those regions do not matter because these states have velocities different from zero and the particle initiates with zero velocity. However, if the particle has to pass through those states to finish the task then the system fails in the task. Thus, it is more difficult to QFCS to learn the vector field due to the unfixed fuzzy sets that introduce more complexity to the problem. Nevertheless, QFCS does not perform too badly. In some parts, QFCS takes more steps to reach the goal than the minimum needed.

Figure 7.24 shows 3 instances of Q-learning from an experiment of 20. As one observes, the learned vector fields are similar to the ones learned by QFCSs. Q-learning learns how to reach the goal in all of the state space xv .

Figure 7.25 shows the average of the obtained results by the QFCSs and the Q-learning system. The first row shows the averaged vector fields learned by the QFCSs and Q-learning. In the second row, there are the number of steps the agent makes to reach the goal from each possible state (x, v) . It is shown how QFCS with the unfixed fuzzy sets has more dark regions than the QFCS with the fixed fuzzy sets, but Q-learning has dark areas also. It means that the problem is not only complex for QFCSs but also

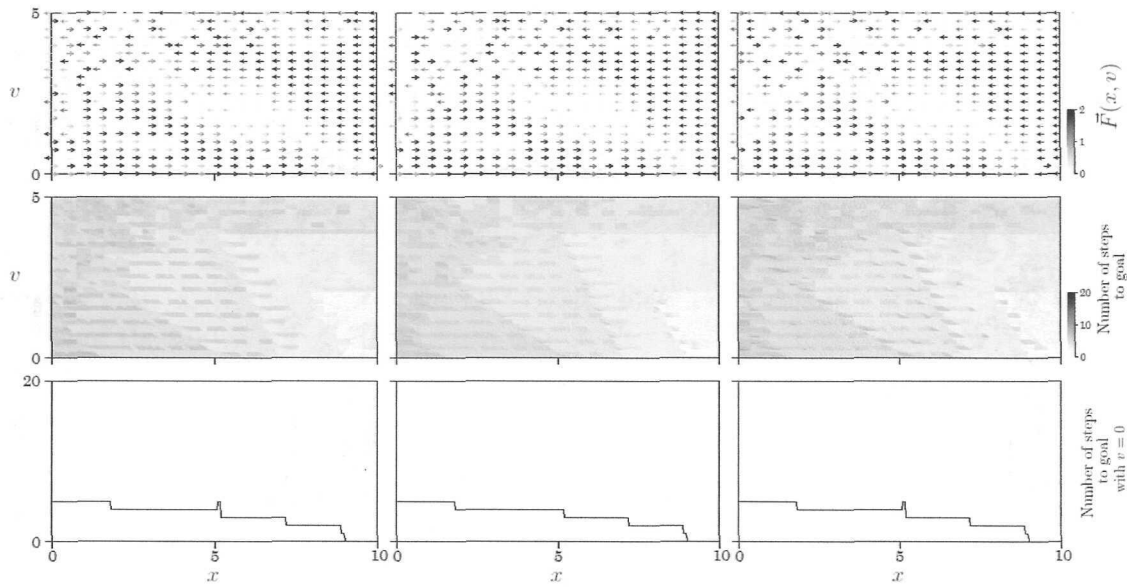


Figure 7.24: Three instances of Q-learning in Particle $_a^{1D}$.

for Q-learning. Figure 7.25.d shows how QFCS with fixed fuzzy set works better than the QFCS with unfixed fuzzy sets. These numbers of steps in the average converge to Q-learning's number of steps. Figure 7.25.e shows how QFCS with the unfixed fuzzy sets converges noisier than the other QFCS, and both of the QFCSs converge similarly to Q-learning.

7.7 The 2-Environment Problem: Particle $_b^{1D}$

Particle $_b^{1D}$ is a 2-Environment problem about the simulation of an inertial particle in one dimension. The goal is to take the particle from a starting position to another through a set of continuous forces. The particle can have negative velocities now. Therefore is more complex than the other one before. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.6. In Q-learning, the axes $x_1 = x$ and $x_2 = v$ of space $x_1x_2a_1 = xvF$ are discretized over 25 intervals and the axes $a_1 = F$ is discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10$ in this problem. This occurs because of the convergence of Q-learning. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to prevent the system from never reaching the goal. This number is much higher than optimal number of steps to reach the goal. When each experiment begins, the $p_{a_1a_2}^l$ in QFCSs and the Q-values of Q-learning are also set up at random in $[0, 10]$. 10 is used because it is the maximum value obtained by reward.

Figure 7.26 shows the vector fields in the initial conditions of the QFCSs and Q-learning. These vector fields are also of one dimension. Therefore, the vector fields are

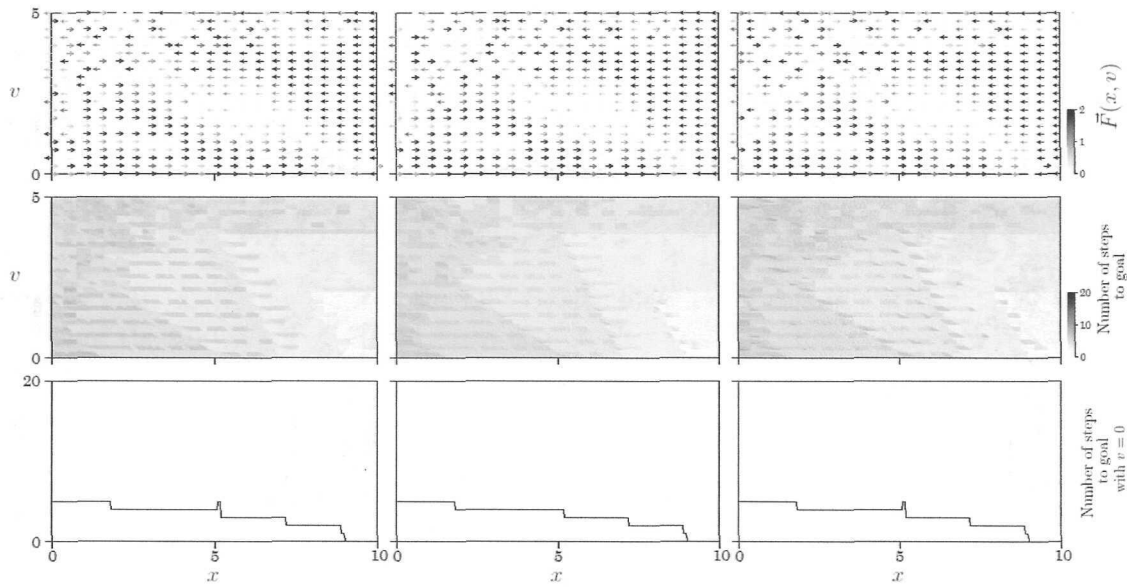


Figure 7.24: Three instances of Q-learning in Particle $_a^{1D}$.

for Q-learning. Figure 7.25.d shows how QFCS with fixed fuzzy set works better than the QFCS with unfixed fuzzy sets. These numbers of steps in the average converge to Q-learning's number of steps. Figure 7.25.e shows how QFCS with the unfixed fuzzy sets converges noisier than the other QFCS, and both of the QFCSs converge similarly to Q-learning.

7.7 The 2-Environment Problem: Particle $_b^{1D}$

Particle $_b^{1D}$ is a 2-Environment problem about the simulation of an inertial particle in one dimension. The goal is to take the particle from a starting position to another through a set of continuous forces. The particle can have negative velocities now. Therefore is more complex than the other one before. The particular parameters used for QFCSs with fixed and unfixed fuzzy sets and Q-learning are shown in Table 7.6. In Q-learning, the axes $x_1 = x$ and $x_2 = v$ of space $x_1x_2a_1 = xvF$ are discretized over 25 intervals and the axes $a_1 = F$ is discretized over 10 intervals. Thus the Q-learning table for the Q-function is of $25 \times 25 \times 10$ in this problem. This occurs because of the convergence of Q-learning. The size of the population is 800 which gives about 50 classifiers per activation region since there are $4 \times 4 = 16$ due to $c = 4$. δ_0 and θ_{GA} were adjusted by experimentation. N_T is high to show convergence of learning. N_S is 200 to prevent the system from never reaching the goal. This number is much higher than optimal number of steps to reach the goal. When each experiment begins, the $p_{a_1a_2}^l$ in QFCSs and the Q-values of Q-learning are also set up at random in $[0, 10]$. 10 is used because it is the maximum value obtained by reward.

Figure 7.26 shows the vector fields in the initial conditions of the QFCSs and Q-learning. These vector fields are also of one dimension. Therefore, the vector fields are

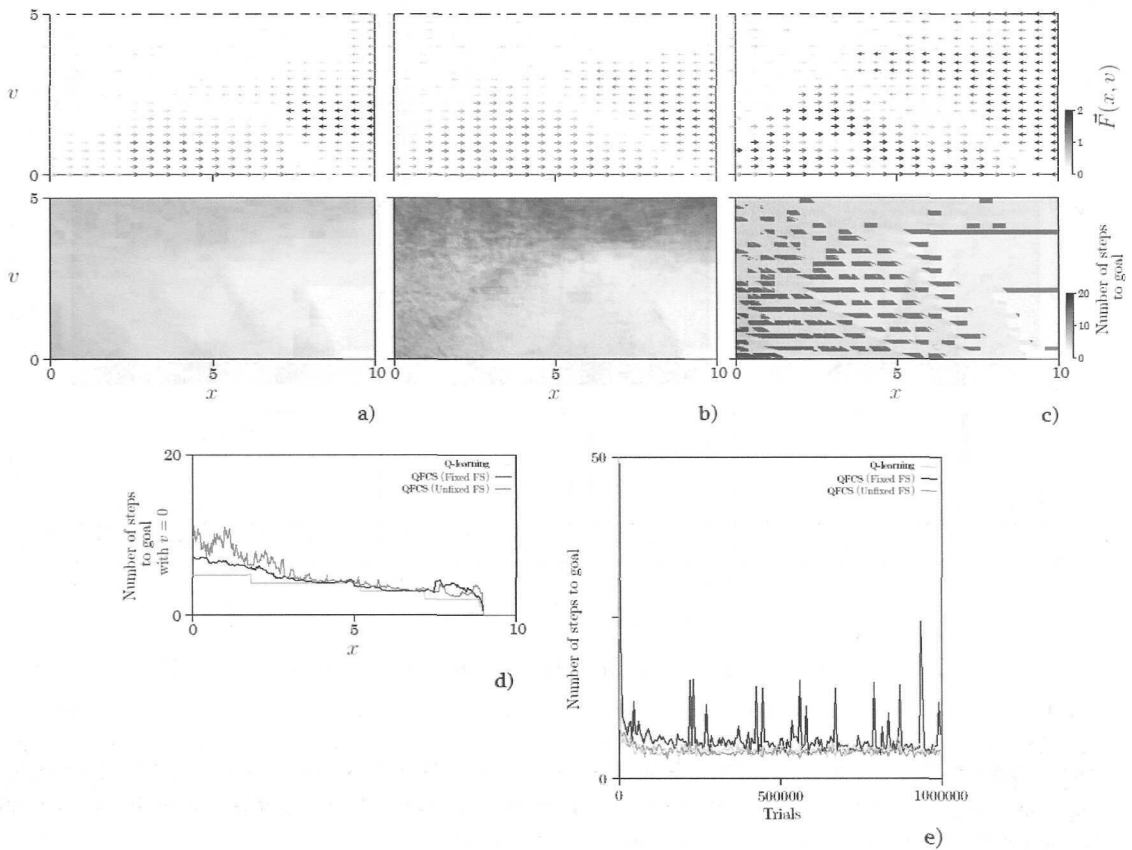


Figure 7.25: a. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with unfixed fuzzy sets. c. Average over 20 runs of the force and number of steps to goal against xv obtained by Q-learning. d. Average over 20 runs of the number of steps to goal with $v = 0$ against x in Particle^{1D}_a. e. Average over 20 runs of the number of steps to goal against trials in Particle^{1D}_a.

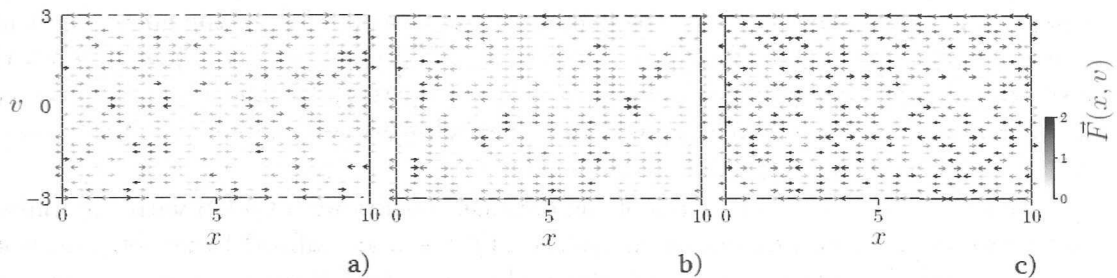


Figure 7.26: a. Initial State of the QFCS with fixed fuzzy sets in Particle^{1D}_b. b. Initial State of the QFCS with unfixed fuzzy sets in Particle^{1D}_b. c. Initial State of Q-learning in Particle^{1D}_b.

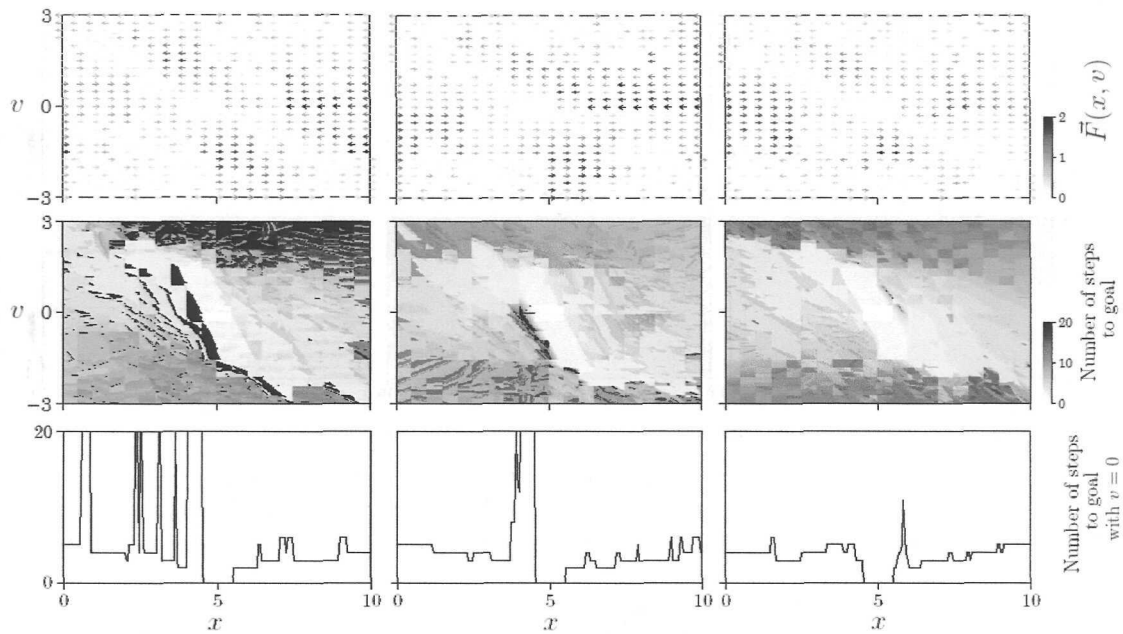


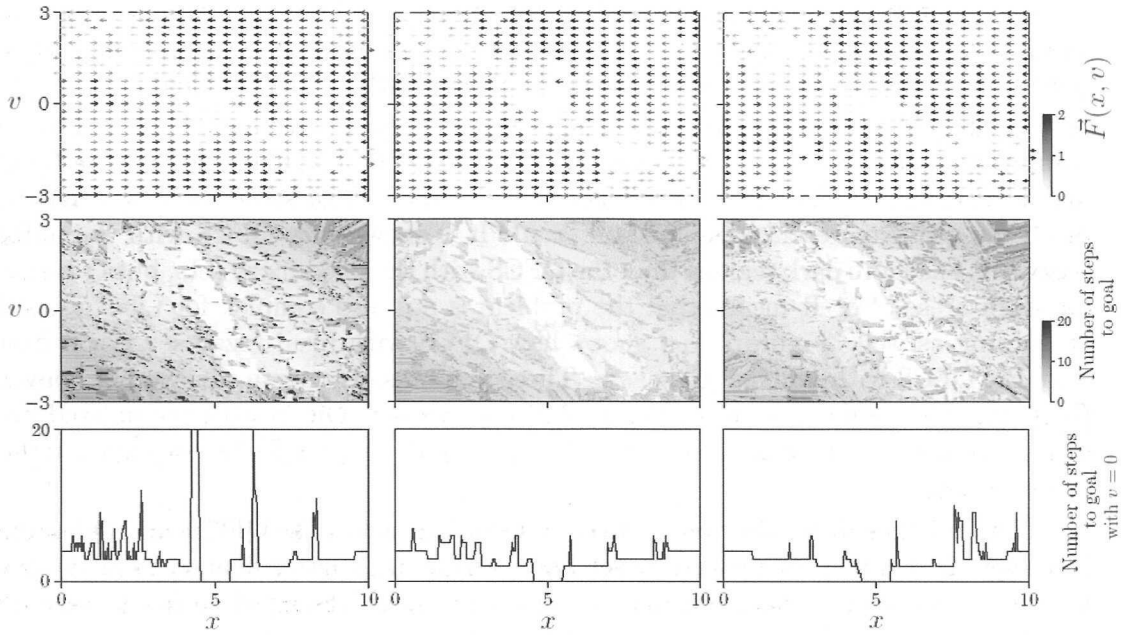
Figure 7.27: Three instances of the QFCS with fixed fuzzy sets in Particle $_b^{1D}$.

also represented as arrows that determine the directions of the vectors and with colors in a gray scale that determine their intensities. These vector fields show how initially the classifiers really represent random vector fields.

Figure 7.27 shows 3 instances of QFCS with fixed fuzzy sets from an experiment with 20 QFCSs. The vector fields are in the first row. The number of steps to reach the goal from each possible state xv are in the second row represented in a gray scale since they are two dimensional functions. The number of steps to reach the goal from each possible state xv with $v = 0$ are in the third row. A number of 20 in the number of steps to goal also means that the system takes 20 steps or more to reach the goal, or that the system simply could not reach the goal at any time. The learned vector field is also noisier, far from the goal and the line with $v = 0$. This is shown by the dark areas in the second row. These areas are in states with $v \neq 0$ in its majority but they are in the line with $v = 0$ in some instances like the one that is in the left of the goal. The third row shows this with more clarity. Close to goal, QFCS does not learn how to reach the goal. But it is not always since there are instances where QFCS has learned. Particle $_b^{1D}$ is more complex than Particle $_a^{1D}$ because of the allowing of negative velocities. QFCS learns well how to reach the goal when it learns.

Figure 7.28 shows 3 instances of the obtained results with QFCS with the unfixed fuzzy sets from an experiment of 20 QFCSs. QFCS with unfixed fuzzy sets can learn a better needed vector field to reach the goal against QFCS with the fixed fuzzy sets. Regions where QFCS does not learn the solution are smaller than with the other QFCS. These regions are similar to small circles. The space xv is also almost learned. This means that, in this case, the freedom of letting the GA learn the activation regions of the classifiers works better.

	Parameter	QFCS		Q-Learning
		Fixed FS	Unfixed FS	
Structure	n	2	2	2
	m	1	1	1
	$[x_1^{\min}, x_1^{\max}]$	[0, 10]	[0, 10]	[0, 10]
	$[x_2^{\min}, x_2^{\max}]$	[-3, 3]	[-3, 3]	[-3, 3]
	$[a_1^{\min}, a_1^{\max}]$	[-2, 2]	[-2, 2]	[-2, 2]
	Δx_1	-	-	0.4
	Δx_2	-	-	0.24
Classifiers (Cl_i)	N	800	800	-
	$[c_{\min}, c_{\max}]$	-	[1, 4]	-
Learning Component	δ_0	1000	2000	-
Discovery Component	θ_{GA}	5000	10000	-
Experiments	N_T	1000000	1000000	1000000
	N_S	200	200	200

Table 7.6: Particular parameters in Particle_B^{1D}.Figure 7.28: Three instances of the QFCS with unfixed fuzzy sets in Particle_B^{1D}.

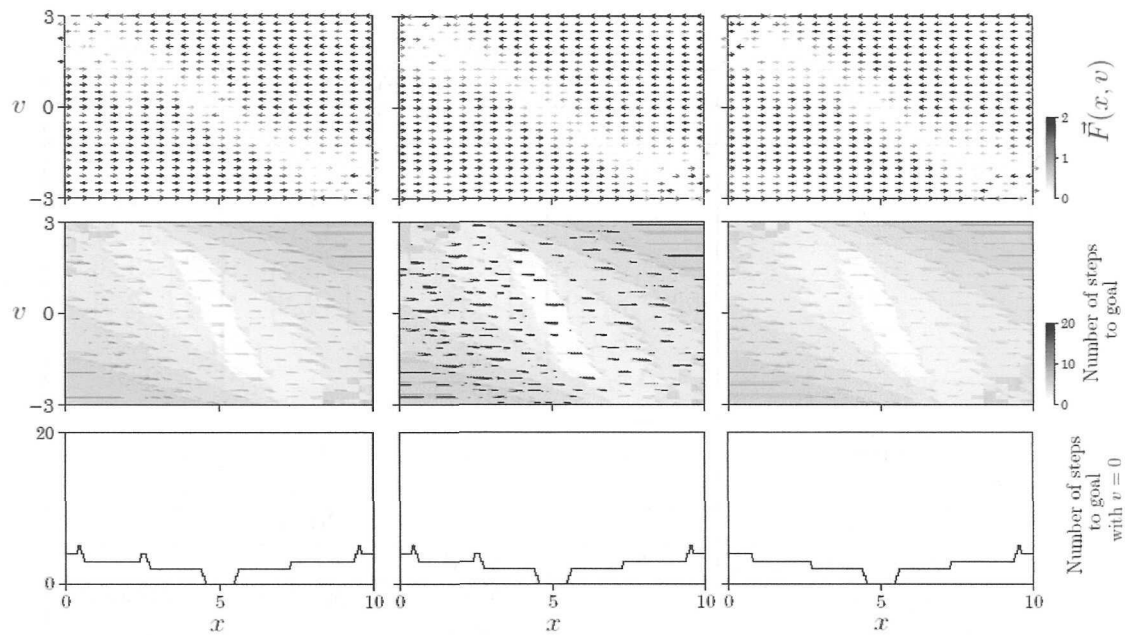


Figure 7.29: Three instances of Q-learning in Particle^{1D}.

Figure 7.29 shows 3 instances of Q-learning from an experiment of 20. As it can be observed, the learned vector fields are similar to the ones learned by QFCSs but less noisy. Q-learning learns how to reach the goal in all of the state space xv . There are some instances where Q-learning has dark regions. These dark region means that Q-learning takes more steps than the normal. This shows that Q-learning also has difficulties with this problem and remarks the complexity of the problem.

Figure 7.30 shows the average of the obtained results by the QFCSs and the Q-learning system. The first row shows the averaged vector fields learned by the QFCSs and Q-learning. In the second row, there are the number of steps the agent makes to reach the goal from each possible state (x, v) . It is shown how QFCS with the unfixed fuzzy sets has more dark regions than the QFCS with the fixed fuzzy sets, but Q-learning has dark areas also. It means that the problem is not also complex for QFCSs but for Q-learning as well. Figure 7.25.d shows how QFCS with fixed fuzzy set works better than the QFCS with unfixed fuzzy sets. These numbers of steps in the average converge to Q-learning's number of steps. Figure 7.25.e shows how QFCS with the unfixed fuzzy sets converges noisier than the other QFCS, and both of the QFCSs converge similarly to Q-learning.

The first row shows the averaged vector fields learned by the QFCSs and Q-learning. Note how QFCS with unfixed fuzzy set has a vector field better defined than the other QFCS. In the second row, there are the number of steps the agent makes to reach the goal from each possible state (x, v) . QFCS with fixed fuzzy sets has more dark areas than the one with the unfixed fuzzy sets and Q-learning. But Q-learning has also more dark areas than QFCSs with unfixed fuzzy sets that are presented as small black lines. Figure 7.30.e shows the average of the number of steps taken initiating with velocity

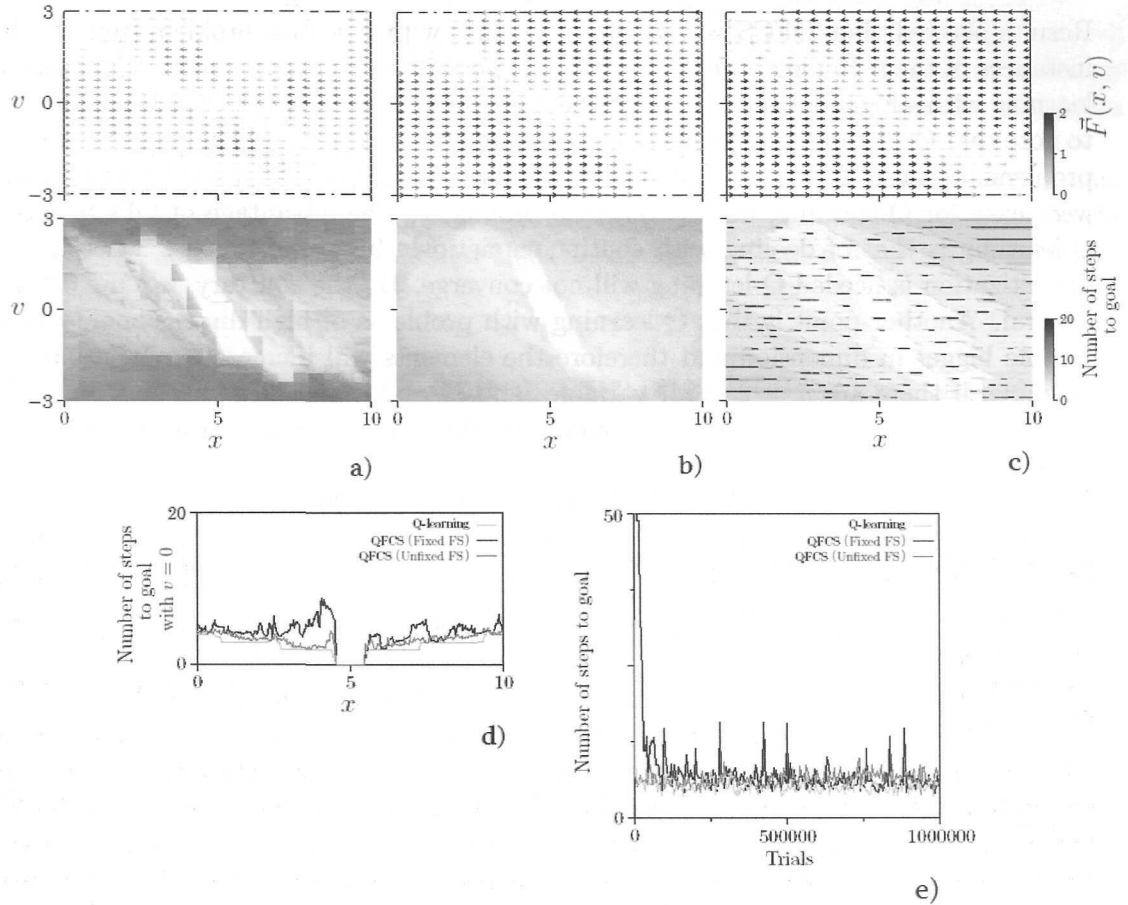


Figure 7.30: a. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with fixed fuzzy sets. b. Average over 20 runs of the force and number of steps to goal against xv obtained by QFCS with fixed fuzzy sets. c. Average over 20 runs of the force and number of steps to goal against xv obtained by Q-learning. d. Average over 20 runs of the number of steps to goal with $v = 0$ against x in Particle_B^{1D}. e. Average over 20 runs of the number of steps to goal against trials in Particle_B^{1D}.

zero. It shows how QFCS with fixed fuzzy set has problems also close to the goal. Finally, Figure 7.30.e shows how QFCS with the fixed fuzzy sets converges noisier than the other QFCS, and both of the QFCSs converge similar to Q-learning.

7.8 General Discussion

Results showed that QFCS is capable of dealing with the frog problem and the five instances of the n -Environment problem. The performance was good compared with Q-learning with a high discretization over the same problems. Consequently, it is important to note that Q-learning cannot have the same high discretization for the two dimensional problems because it does not converge anymore. Furthermore, not all of the problems were easy for Q-learning, like the particle problems. The advantage of QFCS against Q-learning is that of dealing with continuous actions, because in a task where a high discretization is needed Q-learning will not converge. By the contrary, to QFCS this is natural. Another point is that Q-learning with problems of high dimensionality needs a table bigger in dimension and therefore the elements will increase exponentially, for example, if there are $n = 2$ input variables and $m = 2$ input variables and $N_E = 100$ elements of discretization per dimension then the number of elements needed in the table will be

$$(N_E)^{n+m} = 100^{2+2} = 100,000,000 \quad (7.3)$$

while with QFCS, it would be

$$N(d^n) = 800(5^2) = 20,000 \quad (7.4)$$

where N is the number of the population and d is the size per dimension of the prediction hyper-matrices. This makes a huge difference specially with higher dimensions. Thus, QFCS simplifies the learning of the Q-function and it is worthwhile to continue this research in this matter. Another way of representing the Q-values instead of using the hyper-matrices could be by using linear functions given by dot vector products between the input vector and some weight vector as it is done in XCSF. These linear functions approximates the Q-function by hyper-planes. This approach could make QFCS learn by accuracy as XCSF does, and this would reduce drastically the elements needed to approximate the Q-function.

The navigation task in one dimension shows that both QFCSs can follow, with their classifiers, the curvature of the optimal solution. Therefore, the parameter of curvature $c = 4$ in the QFCS with fixed fuzzy sets works well. In the other QFCS, the one with the unfixed fuzzy sets, the GA was able to find out the adequate activation regions to follow the curvature of the optimal solution. This supposition also works well with the navigation in two dimensions. This is suggested by the obtained results where QFCSs learned sub-optimal solutions. Obviously these problems are more complex than the simple navigation task in one dimension since they introduce not only one more dimension in the states but also one more dimension in the actions. Another point is that the introduction of the QFCS with unfixed fuzzy sets was to avoid the problem of having to sintonize the parameter c but the results showed that the introduction of the

zero. It shows how QFCS with fixed fuzzy set has problems also close to the goal. Finally, Figure 7.30.e shows how QFCS with the fixed fuzzy sets converges noisier than the other QFCS, and both of the QFCSs converge similar to Q-learning.

7.8 General Discussion

Results showed that QFCS is capable of dealing with the frog problem and the five instances of the n -Environment problem. The performance was good compared with Q-learning with a high discretization over the same problems. Consequently, it is important to note that Q-learning cannot have the same high discretization for the two dimensional problems because it does not converge anymore. Furthermore, not all of the problems were easy for Q-learning, like the particle problems. The advantage of QFCS against Q-learning is that of dealing with continuous actions, because in a task where a high discretization is needed Q-learning will not converge. By the contrary, to QFCS this is natural. Another point is that Q-learning with problems of high dimensionality needs a table bigger in dimension and therefore the elements will increase exponentially, for example, if there are $n = 2$ input variables and $m = 2$ input variables and $N_E = 100$ elements of discretization per dimension then the number of elements needed in the table will be

$$(N_E)^{n+m} = 100^{2+2} = 100,000,000 \quad (7.3)$$

while with QFCS, it would be

$$N(d^n) = 800(5^2) = 20,000 \quad (7.4)$$

where N is the number of the population and d is the size per dimension of the prediction hyper-matrices. This makes a huge difference specially with higher dimensions. Thus, QFCS simplifies the learning of the Q-function and it is worthwhile to continue this research in this matter. Another way of representing the Q-values instead of using the hyper-matrices could be by using linear functions given by dot vector products between the input vector and some weight vector as it is done in XCSF. These linear functions approximates the Q-function by hyper-planes. This approach could make QFCS learn by accuracy as XCSF does, and this would reduce drastically the elements needed to approximate the Q-function.

The navigation task in one dimension shows that both QFCSs can follow, with their classifiers, the curvature of the optimal solution. Therefore, the parameter of curvature $c = 4$ in the QFCS with fixed fuzzy sets works well. In the other QFCS, the one with the unfixed fuzzy sets, the GA was able to find out the adequate activation regions to follow the curvature of the optimal solution. This supposition also works well with the navigation in two dimensions. This is suggested by the obtained results where QFCSs learned sub-optimal solutions. Obviously these problems are more complex than the simple navigation task in one dimension since they introduce not only one more dimension in the states but also one more dimension in the actions. Another point is that the introduction of the QFCS with unfixed fuzzy sets was to avoid the problem of having to sintonize the parameter c but the results showed that the introduction of the

GA to solve this sintonization, since it is the GA which evolves the activation regions, does not offer any advantages over the manual sintonization.

Parameter $d = 5$ demonstrated to be enough to represent the Q-function according to results of the experiments. Higher values of this makes QFCS not work because there would be many Q-values to learn. Lower values could be not enough to represent the Q-function.

The particle problems were the most difficult ones because of the inertia. However, one QFCS showed to work better in one particle problem and the other QFCS showed to work better in the other particle problem. This means that in this kind of problems there are many subtle details that determine if the GA would evolve well the activation regions or the parameter $c = 4$ would work better.

Another point is that the difficulty of the problem was in the transition function. QFCS does not care which transition function is used, since what it matters is the transition, and it is that transition that makes QFCS have difficulties to solve the problem.

7.9 QFCS against other approaches

It is clear that, similarly to what has been done with Q-learning, a corresponding discretization of the problem could be given to XCS or XCSF. The problem with doing this is that rules in XCS or XCSF relate a set of states of the problem with one action. Therefore, a high discretization would make XCS to have too many rules to be acceptable, specially in the problems where a vector field is required. For example in the navigation task in two dimensions the action was a continuous two dimensional vector. A discretization of $N_E = 100$ elements per dimension would give

$$(N_E)^m = 100^2 = 10,000 \quad (7.5)$$

Therefore, with 10,000 classifiers at least there is one classifier per possible action. It is needed at least more than one classifier per action because XCS learns the complete Q-function. In simple problems where XCS was tested the population was about 2000 classifiers. This means that using XCS or another classifier to compare with QFCS would require a lot of resources. This also shows how continuous problem as the n -Environment need better algorithms that are not based on discretization of the actions like Q-learning, XCS or XCSF. Thus, QFCS does not have that problem and that is an advantage.

The fuzzy classifier like those of Valenzuela's FCS and Parodi and Bonelli's FCS can also deal with the frog problem. This has not been tested yet but in principle they can. The frog problem is simple for these approaches since it is a single step problem. But, for more complex problems like the multi-step ones these FCSs seem not to be capable of dealing with. This is because all of the classifiers form a fuzzy system that represent a function. Thus the function has to change in its forms slowly until it reaches a solution. But it needs reward at each time to make possible this good changes in the form of the function. It is not clear at all if giving only reward in the goal these approaches would work. Therefore, QFCS has that advantage over these FCSs in using

GA to solve this sintonization, since it is the GA which evolves the activation regions, does not offer any advantages over the manual sintonization.

Parameter $d = 5$ demonstrated to be enough to represent the Q-function according to results of the experiments. Higher values of this makes QFCS not work because there would be many Q-values to learn. Lower values could be not enough to represent the Q-function.

The particle problems were the most difficult ones because of the inertia. However, one QFCS showed to work better in one particle problem and the other QFCS showed to work better in the other particle problem. This means that in this kind of problems there are many subtle details that determine if the GA would evolve well the activation regions or the parameter $c = 4$ would work better.

Another point is that the difficulty of the problem was in the transition function. QFCS does not care which transition function is used, since what it matters is the transition, and it is that transition that makes QFCS have difficulties to solve the problem.

7.9 QFCS against other approaches

It is clear that, similarly to what has been done with Q-learning, a corresponding discretization of the problem could be given to XCS or XCSF. The problem with doing this is that rules in XCS or XCSF relate a set of states of the problem with one action. Therefore, a high discretization would make XCS to have too many rules to be acceptable, specially in the problems where a vector field is required. For example in the navigation task in two dimensions the action was a continuous two dimensional vector. A discretization of $N_E = 100$ elements per dimension would give

$$(N_E)^m = 100^2 = 10,000 \quad (7.5)$$

Therefore, with 10,000 classifiers at least there is one classifier per possible action. It is needed at least more than one classifier per action because XCS learns the complete Q-function. In simple problems where XCS was tested the population was about 2000 classifiers. This means that using XCS or another classifier to compare with QFCS would require a lot of resources. This also shows how continuous problem as the n -Environment need better algorithms that are not based on discretization of the actions like Q-learning, XCS or XCSF. Thus, QFCS does not have that problem and that is an advantage.

The fuzzy classifier like those of Valenzuela's FCS and Parodi and Bonelli's FCS can also deal with the frog problem. This has not been tested yet but in principle they can. The frog problem is simple for these approaches since it is a single step problem. But, for more complex problems like the multi-step ones these FCSs seem not to be capable of dealing with. This is because all of the classifiers form a fuzzy system that represent a function. Thus the function has to change in its forms slowly until it reaches a solution. But it needs reward at each time to make possible this good changes in the form of the function. It is not clear at all if giving only reward in the goal these approaches would work. Therefore, QFCS has that advantage over these FCSs in using

Q-learning to learn by reinforcement. This allows QFCS to learn the task only receiving reward in the goal.

Bonarini's approach introduces by the contrary Q-learning. Thus it can model continuous output variables and learning by reinforcement. But the approach is different from the other FCSs. Bonarini's FCSs form sets of classifiers that match different internal fuzzy states. Thus taking one classifier from each set is form a fuzzy system, another selection of the classifiers taken form another fuzzy system and so fourth. In this way, Bonarini's FCS is finding out combinations of fuzzy rules that can form fuzzy system to form the problem. In that manner, there could be many fuzzy systems that solve the problem not only one. This approach has the disadvantage of using only one value per fuzzy rule to approximate the Q-function. This is not good because each rule forms part of many other different fuzzy systems and with those that simple value is not enough. QFCS is finding out fuzzy systems that can solve the problem instead of looking fuzzy rules to form them. With this and with the introduction of a hyper-matrix is capable of learning the Q-function more precisely. Thus, QFCS and Bonarini's FCS solve the problem using many fuzzy systems not only one as the other FCSs. QFCS solved problems where the reward was given only in the goal states while Bonarini's FCS has only been tasted in problems where the reward is given at each time.

7.10 Summary

Experiments were done to test the performance of QFCS. It was used a methodology similar to the common used in LCSs literature. This methodology consists of reporting averaged results. The averages were taken using a sampling of size 20. The important variables to measure are the number of steps the system makes to reach the goal during learning that measures the convergences in learning, the action function learned by the systems that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal. The problems used to test QFCS were:

- THE FROG PROBLEM. This problem was used to prove that QFCS can solve it.
- WORLD_a^{1D} . In this problem both QFCS with the fixed fuzzy sets and QFCS with the unfixed fuzzy set showed to have learned optimal solutions and with a convergence of learning similar to Q-learning.
- WORLD_a^{2D} . In this problem both QFCS with the fixed fuzzy sets and QFCS with the unfixed fuzzy set showed to have learned sub-optimal solutions and with a convergence of learning similar to Q-learning. QFCSs also demonstrated to minimize distance. Thus, the vector fields learned were smoother than the ones obtained by Q-learning.
- WORLD_b^{2D} . QFCS with unfixed fuzzy set shows to have more difficulty in learning the vector field than the other QFCS. Both QFCSs learned sub-optimal solutions.

Q-learning to learn by reinforcement. This allows QFCS to learn the task only receiving reward in the goal.

Bonarini's approach introduces by the contrary Q-learning. Thus it can model continuous output variables and learning by reinforcement. But the approach is different from the other FCSs. Bonarini's FCSs form sets of classifiers that match different internal fuzzy states. Thus taking one classifier from each set is form a fuzzy system, another selection of the classifiers taken form another fuzzy system and so fourth. In this way, Bonarini's FCS is finding out combinations of fuzzy rules that can form fuzzy system to form the problem. In that manner, there could be many fuzzy systems that solve the problem not only one. This approach has the disadvantage of using only one value per fuzzy rule to approximate the Q-function. This is not good because each rule forms part of many other different fuzzy systems and with those that simple value is not enough. QFCS is finding out fuzzy systems that can solve the problem instead of looking fuzzy rules to form them. With this and with the introduction of a hyper-matrix is capable of learning the Q-function more precisely. Thus, QFCS and Bonarini's FCS solve the problem using many fuzzy systems not only one as the other FCSs. QFCS solved problems where the reward was given only in the goal states while Bonarini's FCS has only been tasted in problems where the reward is given at each time.

7.10 Summary

Experiments were done to test the performance of QFCS. It was used a methodology similar to the common used in LCSs literature. This methodology consists of reporting averaged results. The averages were taken using a sampling of size 20. The important variables to measure are the number of steps the system makes to reach the goal during learning that measures the convergences in learning, the action function learned by the systems that shows which action it takes in each possible state, and the number of steps the agent makes from each possible state to reach the goal. The problems used to test QFCS were:

- THE FROG PROBLEM. This problem was used to prove that QFCS can solve it.
- WORLD_a^{1D} . In this problem both QFCS with the fixed fuzzy sets and QFCS with the unfixed fuzzy set showed to have learned optimal solutions and with a convergence of learning similar to Q-learning.
- WORLD_a^{2D} . In this problem both QFCS with the fixed fuzzy sets and QFCS with the unfixed fuzzy set showed to have learned sub-optimal solutions and with a convergence of learning similar to Q-learning. QFCSs also demonstrated to minimize distance. Thus, the vector fields learned were smoother than the ones obtained by Q-learning.
- WORLD_b^{2D} . QFCS with unfixed fuzzy set shows to have more difficulty in learning the vector field than the other QFCS. Both QFCSs learned sub-optimal solutions.

The performance of both QFCSs were similar to Q-learning. QFCSs also demonstrated to minimize distance. The learned vector fields were smoother than the ones obtained by Q-learning.

- PARTICLE_a^{1D} . QFCS with the fixed fuzzy set shows to have better performance than the other one.
- PARTICLE_b^{1D} . QFCS with the unfixed fuzzy set shows to have better performance than the other one.

Chapter 8

Conclusions

This chapter presents a summary of the dissertation with its implications, and the possible future directions to be followed to find better systems to solve multi-step problems by reward with continuous state spaces and with continuous vector action fields.

8.1 Summary

QFCS was introduced as a new fuzzy classifier system. It uses Q-learning and fuzzy logic as its internal representation. It was designed in two versions, one with fixed fuzzy sets and the other one with unfixed fuzzy sets. The second is the generalization of the first. QFCS is capable of solving the n -Environment problem, which was also introduced and is an abstract problem. This n -Environment problem is a multi-step continuous task that is defined through a reward function. This continuous task is defined over a continuous state space, with continuous vector action field, and with an arbitrary transition function. Next is a summary of each QFCS.

- **QFCS with fixed fuzzy sets.** In QFCS, each classifier contains a small fuzzy system (SFS), a matrix containing the expected prediction, and a fixed square sub-region of activation over the input space. The classifiers can only act over their sub-region of activation. In that sub-region of activation, each fuzzy system proposes a continuous vector field as an action by defuzzification. In that way, when an input vector enters the QFCS, classifiers compete to place their actions according to their expected predictions. A Q-learning algorithm is used to learn the task from the environment. This algorithm is used to change the values of the matrices of the expected predictions for each classifier in the QFCS. A GA is applied to evolve rules based on their average expected predictions. This GA evolves only the action parts of the fuzzy systems.
- **QFCS with unfixed fuzzy sets.** QFCS with unfixed fuzzy sets is a generalization of the QFCS with fixed fuzzy sets. Therefore, it works similarly to the one described before but with a few simple modifications. These modifications have something to do specifically with the activation regions of the classifiers that are not fixed any more, with the Learning Component where it is necessary to make

a normalization of prediction values used by the Genetic Algorithm, and with the Genetic Algorithm that is applied now to both the condition and the action parts of the classifiers.

These QFCSs were applied to six different instances of the n -Environment problem called: The Frog Problem, World_a^{1D} , World_a^{2D} , World_b^{2D} , Particle_a^{1D} and Particle_b^{1D} .

- **The Frog Problem** is about a frog that has to jump once to catch a fly. It is a one-step task in a continuous one dimensional state space with a continuous one dimensional action. The Frog Problem is tackled in literature with classifier systems.
- **World_a^{1D}** is a one dimension navigation task. It is a multi-step task in a continuous one dimensional state space with a continuous one dimensional action. The main remark in this problem is that it does not have a function as solution but a region. QFCS is capable of obtaining an optimal solution.
- **World_a^{2D} and World_b^{2D}** are two dimension navigation tasks. The first is without obstacle and the second with them. They are multi-step tasks in continuous two dimensional state spaces with continuous two dimensional action fields. These problems also do not have one optimal solution but many since the solutions are vector regions not a vector functions. QFCS find suboptimal solutions in these problems.
- **Particle_a^{1D} and Particle_b^{1D}** are more realistic problems. They have something to do with the movement of an inertial particle in one dimension. The first simulates a particle that begins from the rest state and can move only in one direction. It can accelerate and desaccelerate and has to reach another region with velocity almost zero. The second is similar but the particle is allowed to move in both directions. These problems are multi-step tasks in continuous two dimensional state spaces with continuous one dimensional actions. QFCS with fixed fuzzy sets showed to work better than the other and vice versa according to the problem.

All of the results obtained with the QFCSs were compared with the ones obtained by Q-learning using high discretizations of the continuous state-action spaces and the continuous action vector fields.

8.2 The Main Essence of QFCS

It was shown how the key ingredient of QFCS is the use of hyper-curves in the state-action space of the problem. These hyper-curves are the domain of the Q-function. The approach is to learn the Q-function over these hyper-curves and evolve them to obtain ones that are in the regions of the state-action space where the Q-function is maximal. To do this, the classifiers incorporate a fuzzy system with a few fuzzy rules and hyper-matrices.

8.3 Implications

The traditional approach used in Fuzzy Classifier Systems (FCS) is changed in QFCS. QFCS does not use fuzzy rules as classifiers but complete fuzzy systems as classifiers. These fuzzy systems are the simplest possible. Thus, it is possible to associate continuous states with continuous actions. This suggests the connection with other possible representations different to fuzzy logic.

The introduction of Q-learning, makes QFCS capable of solving tasks by reinforcement. The Q-values are learned by the classifiers through their prediction matrices. This is also different from the traditional approach because QFCS approximates the Q-values over all of the hyper-surface while in the traditional FCSs one value is learned per fuzzy rule. Classifiers in QFCS compete as in traditional Learning Classifier Systems (LCS) and use an average of the Q-values as strength. This approach is different to the one used in XCSF where the accuracy is used. QFCS have showed to solve multi-step problems where continuous action vector fields are needed while XCSF has not been tested.

Due to the continuous nature of the n -Environment problem, QFCS could be applied to many real life problems, for example: in control problems, because they are defined on continuous state spaces. The advantage of QFCS is to have a system that can learn a task without prior knowledge of the transition function of the task (because the n -Environment problem does not define a particular transition function but an arbitrary one) and that can learn at the same time while acting in the task. Thus QFCS does not matter which transition function the n -Environment is using. Obviously, there are going to be tasks with transitions functions more difficult to QFCS than others, this means that many problems can have the same continuous state-action space and this is all QFCS needs.

8.4 Future Work

QFCS is a first step in the looking for new mechanisms to deal with problems like the n -Environment. QFCS showed that the use of hyper-curves is a good approach for simple continuous problems but with more realistic problems it does not work well. This needs more research. Some ways could be:

- The hyper-curves could be changed by other forms not only by Fuzzy Logic Systems. Using the one used in XCSF [15] is faster when learning and acting since they are hyper-planes. These hyper-planes would have the next form per classifier Cl_i

$$a_i = c_{i1}x_1 + \dots + c_{in}x_n + b_i, \quad (8.1)$$

where c_{ij} and b_i are constants and $j \in \{1, \dots, n\}$.

- Replacing the matrices by hyper-planes as in XCSF [15] and using a similar concept that the accuracy could work. In this case, the classifiers will not evolve over those regions where the Q-function is maximal any more. Instead, the Genetic Algorithm

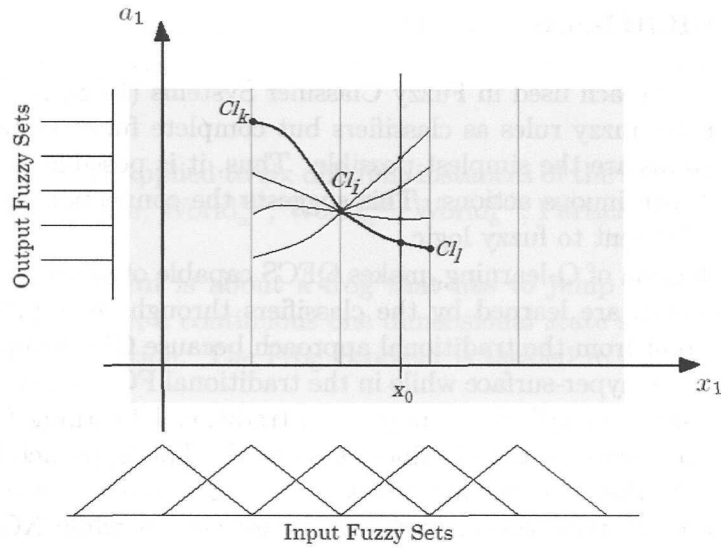


Figure 8.1: Curves generated by one classifier in combination with all of its close classifiers

(GA) evolves classifiers that represent the Q-values over all of the state-action space. These prediction functions would have the next form per classifier Cl_i

$$q_i = c_{i1}u_1 + \dots + c_{in}u_n + b_i, \quad (8.2)$$

where c_{ij} and b_i are constants and the variables u_j represent a parametrization of the hyper-surfaces of the fuzzy systems over the activation region as follows

$$a_i = a_i(x_1(u_1), \dots, x_n(u_n)). \quad (8.3)$$

- Following the XCSF introduced in [28], the possible way would be to associate a hyper-plane over the state-action space as the prediction of Q-values and use two GAs, one to evolve classifiers and another one to evolve actions. This prediction function would have the next form per classifier Cl_i over the activation region and all of the action space

$$q_i = [c_{i1}x_1 + \dots + c_{in}x_n] + [d_{i1}a_1 + \dots + d_{im}a_m] + b_i, \quad (8.4)$$

where c_{ij} , d_{i1} and b_i are constants.

- It would be possible to make a new FCS where the classifiers are fuzzy rules and cooperate based on QFCS. This approach is similar to Bonarini's FCS but without the competition among fuzzy rules. Figure 8.1 shows the curves formed by one classifier Cl_i with all of its neighbors. Each curve has to have a matrix prediction as in QFCS. These matrices will be stored out of classifiers. The decision of taking an action is similar than in QFCS. It takes the action that has the maximum prediction value in the prediction matrix corresponding to the intersections between

the input x_0 and the curves formed by all of the matched classifiers. The fitness F_i value of the classifier Cl_i will be

$$F_i = \max\{\bar{q}_{ikl}\}, \quad (8.5)$$

where \bar{q}_{ikl} is the average of the prediction values over the curve formed by the classifiers Cl_k , Cl_i and Cl_l .

Bibliography

- [1] ALPAYDIN, E. *Introduction to Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2004.
- [2] BISHOP, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [3] BONARINI, A. Evolutionary learning of fuzzy rules: Competition and cooperation. *Fuzzy Modeling: Paradigms and Practice* (1996), 265–284.
- [4] BONARINI, A., BONACINA, C., AND MATTEUCCI, M. Fuzzy and crisp representations of real-valued input for learning classifier systems. In *Proceedings of IW LCS99* (1999), pp. 228–235.
- [5] BULL, L., Ed. *Applications of Learning Classifier Systems*. Springer, 2004.
- [6] BULL, L. *Learning Classifier Systems in Data Mining (Studies in Computational Intelligence)* (*Studies in Computational Intelligence*). Springer, 2008.
- [7] BUTZ, M. V. *Anticipatory Learning Classifier Systems*. Springer, 2002.
- [8] DAM, H. H., ABBASS, H. A., AND LOKAN, C. Be real! XCS with continuous valued inputs. In *Genetic and Evolutionary Computation Conference* (2005), pp. 85–87.
- [9] DRUGOWITSCH, J. *Design and Analysis of Learning Classifier Systems: A Probabilistic Approach (Studies in Computational Intelligence)*. Springer, 2008.
- [10] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. Springer, 2007.
- [11] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.
- [12] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1992, 1975.
- [13] HOLLAND, J. H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *Machine Learning: An Artificial Intelligence Approach 2* (1986).

- [14] HOLLAND, J. H., HOLYOAK, K. J., NISBETT, R. E., AND THAGARD, P. R. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986.
- [15] LANZI, P. L., LOIACONO, D., WILSON, S. W., AND GOLDBERG, D. XCS with computable prediction in continuous multistep environments. ILLiGAL Report 2005018, Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, May 2005.
- [16] LANZI, P. L., STOLZMANN, W., AND WILSON, S. W., Eds. *Advances in Learning Classifier Systems*. Springer, 2008.
- [17] LANZI, P. L., AND WILSON, S. W. Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation* 8, 4 (2000), 393–418.
- [18] MATTEUCCI, M. Learning fuzzy classifier systems: Architecture and explorations, May 2000. <http://citeseer.ist.psu.edu/matteucci00learning.html>.
- [19] OGATA, K. *Modern Control Engineering 4a. Edition*. Prentice Hall, 2003.
- [20] PARODI, A., AND BONELLI, P. A new approach to fuzzy classifier system. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (1993), pp. 223–230.
- [21] RAMÍREZ-RUIZ, J. A., VALENZUELA-RENDÓN, M., AND TERASHIMA-MARÍN, H. A new approach to Fuzzy LCSs in two-dimensional continuous multistep environment with continuous vector actions. *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation GECCO 2008* (2008), 1433–1434.
- [22] RAMÍREZ-RUIZ, J. A., VALENZUELA-RENDÓN, M., AND TERASHIMA-MARÍN, H. QFCS: A fuzzy LCS in continuous multi-step environments with continuous vector actions. *Parallel Problem Solving from Nature PPSN 2008 5199* (2008), 286–295.
- [23] RAMÍREZ-RUIZ, J. A., VALENZUELA-RENDÓN, M., AND TERASHIMA-MARÍN, H. uQFCS: QFCS with unfixed fuzzy sets in continuous multi-step environments with continuous vector actions. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation GECCO 2009* (2009).
- [24] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Ribera del Loira, 28. 28042 Madrid (Spain), 2004.
- [25] STOLZMANN, C. W. Learning classifier systems using the cognitive mechanism of anticipatory behavioral control. Tech. Rep. 4, First European Workshop on Cognitive Modeling, November 1996.
- [26] STONE, C., AND BULL, L. For real! XCS with continuous valued inputs. *Evolutionary Computation* 11, 3 (2003), 299–336.

- [27] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction*. MIT Press, London, England, 1998.
- [28] TRUNG HAU TRAN, CDRIC SANZA, Y. D., AND NGUYEN, D. T. XCSF with computed continuous action. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (2007), pp. 1861–1869.
- [29] VALENZUELA-RENDÓN, M. The fuzzy classifier system: Motivations and first results. *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature PPSN-1 496* (1990), 338–342.
- [30] VALENZUELA-RENDÓN, M. The fuzzy classifier system: A classifier system for continuously varying variables. In *Proceedings of the Fourth International Conference in Genetic Algorithms* (1991), pp. 346–353.
- [31] VALENZUELA-RENDÓN, M. Reinforcement learning in the fuzzy classifier system. Research Report CIA-RI-031, Center for Intelligent systems, Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, N. L., México, January 1997.
- [32] WANG, L.-X. *A Course in Fuzzy Systems and Control*. Prentice Hall, Upper Saddle River, NJ 07458, 1996.
- [33] WILSON, S. W. Classifier fitness based on accuracy. *Evolutionary Computation* 3, 2 (1994), 1–44.
- [34] WILSON, S. W. Function approximation with a classifier system. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2001)* (July 2001), vol. 4, pp. 974–981.
- [35] WILSON, S. W. Get real! XCS with continuous valued inputs. In *Learning Classifier Systems: From Foundations to Applications* (February 2004), vol. 1813/2000 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 209.
- [36] WILSON, S. W. Three architectures for continuous action. In *Lecture Notes in Computer Science* (2007), vol. 4399, Springer-Verlag, pp. 239–257.
- [37] ZADEH, L. A. Fuzzy sets. *Information and Control* 8 (1965), 338–353.

