Thèse de Doctorat de l'Université Paris-Sud

Specialité: Informatique

Présentée per
Carlos Kavka

pour obtenir le grade de
Docteur de l'Université Paris-Sud

EVOLUTIONARY DESIGN OF GEOMETRIC-BASED FUZZY SYSTEMS

Soutenue le 6 Juillet 2006 devant le jury composé de

Cyril Fonlupt	Professeur à l'Université du Littoral, examinateur
Laurent Foulloy	Professeur à l'Université de Savoie, rapporteur
Jean-Sylvain Lienard	Directeur de Recherche CNRS, examinateur
Zbigniew Michalewicz	Professeur à l'Université d'Adelaide, rapporteur
Marc Schoenauer	Directeur de Recherche INRIA, directeur de thèse

Contents

Introduction

Fuzzy systems are a technique that allows the definition of input-output mappings based on linguistic terms of every day common use. This approach has been particularly successful in the area of control problems. In these problems, the objective is to define a mapping (controller) that produces the correct outputs (control signals) given a set of inputs (measurements).

Traditional approaches in control, of which the proportional integral derivative (PID) control method is maybe the best representative, define the controller as a parameterized mathematical function and then determine the optimum values for its set of parameters in order to approximate the desired mapping.

Fuzzy systems use a different approach. The first step in the definition of a fuzzy system is the partition of the input and output domains in a set of linguistic terms (or linguistic labels) defined in *natural* language. These terms, which are imprecise by their own linguistic nature, are however defined very precisely by using fuzzy theory concepts. As a second step, a set of rules is defined in order to associate inputs with outputs. As an example, let us consider a typical vehicle control problem where the objective is to define a controller that determines the pressure to be applied to the break pedal, based on the vehicle speed and the distance to an obstacle. The two inputs are the speed of the vehicle and the distance to the obstacle, and the single output is the break pressure. The input variable *speed* can be partitioned in the three linguistic terms *low*, *medium* and *high*, which refer to the concepts of the vehicle moving at low speed, medium speed and high speed respectively. In the same way, the input variable *distance* is partitioned in the three linguistic terms *near*, *medium* and *far*. The output variable *pressure* can be partitioned in *low*, *medium* and *strong*. Based on these linguistic terms, a set of rules that establish the correspondence between inputs and outputs can be defined. For example, one of these rules can establish that the pressure on the break pedal should be *strong* if the speed is *high* and the distance to the obstacle is *near*. In principle, a set of nine rules has to be defined in order to determine the action that must be performed by considering all combinations of speed and distance linguistic terms.

The fuzzy system is evaluated by determining the degree of application of all rules to the actual situation, and then computing the output of the system based on the values determined by the consequents of the rules, weighted by their corresponding degree of applicability. In this way, the rules that are more adequate to the actual situation have a larger weight (or higher responsibility) in the computation of the output value.

As it can be appreciated, the method is simple and has an interesting intuitive appeal. However, the method presents two main disadvantages:

- The number of rules grows exponentially with the number of input variables and the number of linguistic terms used in their definition.
- The selection of adequate parameters for the definition of linguistic terms is a complex process and becomes more complicated when the number of input variables grows.

To overcome these difficulties, several techniques have been proposed. Since from a strictly mathematical point of view, a fuzzy system in its base form is a parameterized system which can be analitically represented, optimization methods that perform parameter learning, like gradient descent or evolutionary algorithms, have been widely used. Gradient descent based methods, like the back-propagation algorithm, have been successfully applied in many cases. However, one of their main disadvantages is the fact that the optimization procedure can be trapped in local minima. Evolutionary algorithms, which perform also an optimization procedure, have demonstrated to be more robust facing this problem.

Evolutionary algorithms are an optimization technique based on ideas from the evolution of species of Darwin, which maintain a population of alternative solutions to a particular problem. Each alternative solution (individual) is evaluated on the problem, and its quality is measured and established as a single real value (fitness). This value is used to select some individuals from the population which are considered as candidates to be involved in the definition of the next generation population. In the most usual evolutionary approaches, the selected individuals undergo mutation and crossover, two operations defined in probabilistic terms. This process continues till individuals with adequate quality are obtained or other termination criterium is reached.

Usually, evolutionary algorithms are applied to the definition of fuzzy systems by considering individuals that represent the partition of the input and output space, and in some cases the definition of the rules. The fitness computation is performed by evaluating the fuzzy system represented by the individuals on the target problem. The evolutionary operations normally update the definition of linguistic terms, and update, add or remove rules. In some cases, depending on the representation and the definition of the operators, the evolutionary algorithm can produce fuzzy systems with a restricted semantic interpretability of linguistic terms and rules.

Thesis plan

The main contribution of this thesis is the definition of a new model of fuzzy system where the exponential growth of the number of rules with respect to the number of input variables is reduced, with an efficient representation for the design, using evolutionary algorithms. In the proposed model, the partition of the input space is not defined as a regular structure built as the intersection of the linguistic labels of input variables, as usual in fuzzy systems, but in terms of multidimensional regions, each one associated with a single fuzzy rule.

The partition is defined based on well known concepts of computational geometry: the Voronoi diagrams and the Delaunay triangulations. The fuzzy system defined in terms of this partition has a clear and appealing structure. The representation of the individuals for evolutionary algorithms is simple, since each region in the multidimensional input space is represented with a single point. This geometric representation allows the use of geometric based operators for evolution. As an added advantage, the model allows an interesting approach for

the inclusion of *a priori* knowledge about the solution of the problem in the individuals before and during the evolution.

The thesis report is organized in 5 chapters. The first two chapters are introductory and provide the basic concepts of evolutionary algorithms and fuzzy systems. The first chapter introduces evolutionary algorithms with a presentation of the most important branches of evolutionary computation and a discussion about their main elements: representation, selection and operators. The second chapter presents the basic concepts of fuzzy systems, together with information on their use in control problems and a description of the most representative current models. A discussion on partition of input domain characteristics of the most important models is also included, together with information on evolutionary approaches used to evolve fuzzy systems. The chapter also includes a presentation of recurrent fuzzy systems, an extension that allows the application of fuzzy systems to temporal problems. Current recurrent models and their main properties are also presented.

Chapter 3 presents the *Voronoi approximation*, a method proposed to provide data approximation based on computational geometry concepts. The chapter introduces basic concepts and techniques like Voronoi diagrams, Delaunay triangulations and barycentric coordinates. The chapter also presents a representation for evolutionary algorithms and the associated operator definitions. Included is a study of the effect of different values of crossover and mutation probabilities on a function approximation problem solved by using the representation and operators proposed.

Chapter 4 presents a new method to define fuzzy systems in terms of the Voronoi approximation, with a detailed study of their properties. The evolutionary approach to evolve fuzzy systems is discussed, considering also the possibility to include *a priori* knowledge before the evolution. Experimental results on evolutionary design of Voronoi based fuzzy systems are presented in two control problems: an inverted cart pole system and a typical robot control application.

Chapter 5 presents the definition of recurrent fuzzy systems based on the Voronoi approximation, including the proposed evolutionary approach and a discussion of their properties. The evolutionary approach to design recurrent Voronoi-based fuzzy systems is evaluated in two problems: a system identification problem, where the outputs are defined in terms of past inputs and outputs and a problem from evolutionary robotics, where the ability to introduce *a priori* knowledge in the form of recursive rules is demonstrated.

Some chapters present experimental results, with an indication of average running CPU time. Except where noted otherwise, running times are expressed in seconds and correspond to executions on a Linux computer system with an Intel Pentium 4 processor, running at 2.8 GHz.

Chapter 1

Evolutionary Computation

This chapter presents a brief overview of evolutionary computation, its main characteristics and the most important instances. The objective of the chapter is to provide a context for the research performed in this thesis, with special emphasis on the main theoretical and practical concepts.

The chapter is organized as follows. Section 1.1 provides an introduction to evolutionary computation. The main instances of evolutionary algorithms are described in section 1.2. Section 1.3 discusses the most usual representations, and sections 1.4 and 1.5 introduce details on the operators commonly used in evolutionary algorithms.

1.1 Overview

Evolutionary Computation (EC) is the branch of soft computing that includes the stochastic optimization algorithms called Evolutionary Algorithms (EA) [43, 32]. This kind of algorithms can deal with search, optimization and machine learning problems applying ideas from the natural Darwinian evolution [49]. Evolutionary algorithms exhibit robust performance and global search characteristics, requiring only a simple quality measure from the environment.

Evolution is the result of the interplay between the creation of new genetic information and its evaluation and selection [11]. An evolutionary algorithm maintains a population of individuals that represent potential solutions to a given problem [9]. Each individual is evaluated in its environment, producing a quality measure usually called *fitness*. New individuals are obtained from the original population by applying variation operators, with the most important usually called recombination and mutation. Recombination consists of an interchange of information between individuals and mutation corresponds to a stochastic transformation of a single individual [42]. The evolutionary process tends to favor individuals with higher fitness to be selected for recombination.

More formally, the evolutionary algorithm consists of a loop that performs the operations of mutation, recombination and selection, as shown in the algorithm 1 (based on [9] and [137]). The parameters $\mu, \theta_\nu, p_c, p_m, \theta_{s1}, \theta_{s2}$ are generic parameters. Each operator might have additional parameters for specific purposes.

The population P at time $t = 0$ with size μ is initialized (line 3) and all its individuals

1: **procedure** EVOLUTION($\mu, \theta_\nu, p_c, p_m, \theta_{s1}, \theta_{s2}$)
2: $t \leftarrow 0$
3: $P(t) \leftarrow initialize(\mu)$
4: $F(t) \leftarrow evaluate(P(t))$
5: **while** $\nu(P(t), \theta_\nu) \neq true$ **do**
6: $P'(t) \leftarrow select\_parents(P(t), \theta_{s1})$
7: $P''(t) \leftarrow recombine(P'(t), p_c)$
8: $P'''(t) \leftarrow mutate(P''(t), p_m)$
9: $F'''(t) \leftarrow evaluate(P'''(t))$
10: $P(t+1) \leftarrow select(P'''(t) \cup P(t), F(t) \cup F'''(t), \theta_{s2})$
11: $t \leftarrow t+1$
12: **end while**
13: **end procedure**

Algorithm 1: Pseudocode of a general evolutionary algorithm.

are evaluated (line 4). The loop (parental selection, recombination and mutation, survival selection) is repeated while the termination condition ν with parameter θ_ν is not fulfilled (line 5). A set of parents $P'(t)$ is selected from the total population with a parameter θ_{s1} (line 6). The population $P''(t)$ is obtained by recombining individuals from $P'(t)$ (line 7), operation that is performed with a given probability p_c. The mutation operation is applied with probability p_m (line 8) to the elements of $P''(t)$ and a new population $P'''(t)$ is obtained. It is evaluated (line 9), and the population of next generation $P(t+1)$ is built by selecting individuals from $P'''(t) \cup P(t)$ (line 10) based on their fitness values, and the parameter θ_{s2}.

1.2 History

There are four main dialects in Evolutionary Computation: *genetic algorithms*, *evolution strategies*, *evolutionary programming* and *genetic programming* [42]. The first three were developed independently, but all share a number of common properties. The main characteristics of these four main branches are described below, with more details on their implementation in following sections.

1.2.1 Genetic algorithms

Genetic algorithms (GA) were proposed by Holland in 1962, with large further contributions by Davis, Goldberg [54] and De Jong [70]. Individuals are usually represented as bit strings, however genetic algorithms using other representations were also proposed. The main variation operator is recombination, with low emphasis on mutation, which is usually considered as a kind of background operator [49]. Selection is implemented in probabilistic terms. In the original proposal by Holland, the probability of an individual to be selected for recombination is proportional to its observed performance.

The behavior of genetic algorithms is usually analyzed in terms of the *schema* theory [54, 44]. Assuming a binary representation, a schema is a similarity pattern that denotes a subset

of $\{0,1\}^l$, which represents a useful structure used as a *building block* for the genetic algorithm in order to define the solution. The *building block hypothesis* establishes that the GA operates by combining small highly fit building blocks in order to get larger building blocks [44]. The intuitive idea is that parents with well performing building blocks (good characteristics) have a non-zero probability to produce children that will perform even better, thus implicitely assuming some kind of linearity of the fitness function.

1.2.2 Evolution strategies

Evolution strategies (ES) were introduced by Rechenberg and Schwefel in 1965 [125, 130], as a tool to solve difficult real valued parameter optimization problems. In the original implementation, an individual consists of a vector of discretized real numbers, which is mutated in every generation, by adding to it a normally distributed random vector [58]. The model was enhanced to include a population of multiple individuals, supporting also recombination operations. Selection is usually deterministic and is performed after applying the variational operators (no parental selection per se), in order to determine the children to be selected for the next generation [106]. It is a usual approach in ES to use self-adaptation for the value of the deviation of the mutation operation [40].

A standard notation to classify ES was introduced by Schwefel [132]. By using this notation, a $(\mu + \lambda)$-ES is an evolution strategy that generates λ children from μ parents, with the best μ individuals selected for the next generation, from the set of size $\mu + \lambda$ defined by both parents and children. An (μ, λ)-ES is an evolution strategy that also generates λ children from μ parents, but selecting the μ best individuals just from the λ children.

1.2.3 Evolutionary programming

Evolutionary programming (EP) was proposed by Fogel in 1962 [123], being originally developed to evolve finite state machines [106]. Strong emphasis is placed on mutation operators, not including recombination operators at all. Selection is used like in ES to select the children for the next generation [106], but is implemented in probabilistic terms [12].

1.2.4 Genetic programming

Genetic Programming (GP) was proposed by Cramer in 1985 [30], but was really developed and popularized by Koza [90]. Some authors consider it to be a sub-branch of GA, however, it defines conceptually a different approach to evolutionary computation [86]. The objective of GP is to build a computer program that can solve a particular task. For example, GP has been used to solve function approximation and classification problems [119]. In the evolutionary process, each individual represents a computer program. The evaluation of an individual corresponds to the execution of the program in a specific environment. A standard representation for a program is a tree structure defined in a language like LISP. Recombination is the most important operator, and usually consists of interchanging substructures from both parents, with self adaptation of the parameters also possible [5] though not widely used. In the first implementations from Koza, mutation was not used at all, however in current

implementations, mutation is usually implemented as either the insertion or the deletion of program substructures, or the change of a node operator.

1.3 Representations

The representation (or genotype) encodes the characteristics of interest of the potential solutions (usually called phenotype). The efficiency of the search procedure heavily depends on the characteristics of the selected representation [33] and the adequacy of the associated variation operators [44]. The most used representations are bit strings, vectors of real numbers, permutations, parse trees, production rules. But one of the strength of Evolutionary Algorithms comes from their ability to also handle problem-specific representations, as will be demonstrated in this thesis. The main characteristics of the usual representations listed above are defined below.

1.3.1 Bit strings

Binary strings of fixed length are the standard representation for basic or *canonical* genetic algorithms. An individual I of length l is defined as the vector $I = (b_1 \dots b_l)$, where $b_i \in \{0, 1\}, (1 \leq i \leq l)$. Bit strings can encode directly the values of boolean variables that are associated to boolean optimization problems, like for example the knapsack problem [85]. Problems that involve integer variables can be represented by assigning a substring of n bits to each variable, where n bits are enough to represent the expected range of values. This representation implies that binary values have to be converted to integers every time the individual has to be evaluated. (Discretized) real numbers can also be represented, by assigning to each variable a range of bits. The number of bits assigned determines the range and the precision of the real values represented. Again, conversion has to be performed before evaluation.

Binary strings were used initially in most applications of genetic algorithms mainly because the schema theory establishes that the number of schema available is maximized by using binary variables [9], a theoretically desirable property, even though this argument has been criticized [6]. However, most recent results showed that problem-specific representations perform in general better (if ad-hoc variation operators are also provided): for instance, it is now agreed that real valued representations should be preferred in the case of real variables [104].

1.3.2 Vectors of real numbers

The state of art of evolutionary computation when dealing with real variables is to use real valued based representations. In this representation, an individual I of length l is defined as the vector $I = (a_1, \dots, a_l)$, where $a_i \in [low_i \dots high_i], low_i < high_i, low_i \in \mathbb{R}, high_i \in \mathbb{R}, (1 \leq i \leq l)$. A vector of real numbers is the most direct representation for problems where the variables can take real values, like continuous optimization problems. The representation is very efficient because there is no decoding stage, since the values of the parameters correspond to the values of the variables.

Vectors of real numbers are the standard representation for evolution strategies and evolutionary programming. However, this representation has been used widely in other contexts, for example, in constrained parameter optimization problems [110] and neural network evolution [151, 152].

1.3.3 Permutations

The permutation representation [146] is a natural representation for many classical optimization problems like the traveling salesman or scheduling problems. Each individual represents an arrangement of elements, usually encoded with integer values in a linear or matrix structure.

However, it is interesting to note that even when the phenotype is a permutation, some other representations might be better suited for evolution [141].

1.3.4 Parse trees

Parse tree representation is the main characteristic of genetic programming. The basic idea consists in representing a computer program by using a tree, where the internal nodes correspond to operations and the leaves to data objects (constants and variables). In most applications, programs are represented with a LISP like syntax [90], which is a good candidate since its structure can be mapped directly into a tree. The strength of parse tree representation comes from its closure with respect to the crossover – exchange of two randomly chosen sub-trees of the parent trees: the resulting tree will always represent a valid program.

1.3.5 Production rules

Evolutionary algorithms have been also applied to the design of rule learning systems, and the resulting branch of EC is called "Classifier systems" [93]. Two main approaches have been followed to represent rule sets: the Michigan approach and the Pittsburgh approach [50].

The Michigan approach

In the Michigan approach, each individual encodes a single rule, meaning that in order to evaluate a rule system, a set of individuals has to be selected. Evaluation is complex, since a rule set has to be built before the evaluation process. After evaluation, the global fitness has to be distributed to the individual rules that conform the evaluated set.

One of the first representations based on rules following the Michigan approach was the classifier system from Holland [71]. Each individual represents a single rule, specifying the condition and the action section with a fixed length string defined with characters in the set $\{0, 1, *\}$. For example, the following string:

$$\begin{array}{cc} \mathit{condition} & \mathit{action} \\ 01*11**1 & 0100 \end{array} \tag{1.1}$$

encodes a rule with eight binary inputs and four binary outputs. This rule matches messages (in Holland terminology) specified by the pattern $01*11**1$ where $*$ is a wildcard character

symbol that can match a 0 or a 1. Rules that match a message produce the output message determined by its action. During the evolution, the strength of the rules (a classifier performance statistic) is determined by using a bidding mechanism known as the *bucket brigade* algorithm [148]. An extension of standard classifier systems is the XCS model proposed by Wilson [147], which is based on Q-learning. In XCS, the fitness of a classifier is defined based on the accuracy of its prediction and not on the prediction itself.

In the Michigan approach, there is no convergence in the population unless a single rule can solve the problem, which is not usually the case. The evolutionary process needs to co-evolve a population of cooperative rules that can jointly solve the problem [86].

The Pittsburgh approach

In the Pittsburgh approach, each individual encodes a complete rule set. Usually the number of rules is not fixed, requiring a variable length representation. Compared with the Michigan approach, the representation can be more complex and possibly algorithms that limit the growing of rule sets have to be applied.

The GABIL system proposed by De Jong [70, 72] follows the Pittsburgh approach, with each individual representing a set of rules in Disjunctive Normal Form (DNF). The k possible values (features) of a variable are represented with k bits, indicating with a 1 or 0 that a particular feature is allowed or not respectively. The consequent of the rule is represented with a binary string that indicates the class in which the input pattern has to be classified. The individual is a variable length binary string that represents an unordered set of fixed length rules.

The GIL model proposed by Janikow [69] uses a similar approach in terms of representation, however, the kind of operators defined are different.

In the SAMUEL model proposed by Grefenstette [55], each individual also represents a complete rule set. The main difference when compared with the GIL model, is that the rules are represented symbolically rather than with a binary pattern. The left hand side of each rule specifies the set of conditions for the input variables and the right hand side the set of recommended actions, together with their corresponding strength. Conditions for input variables can be specified in terms of ranges with upper and lower bounds, or with a list of values. The strength specifies the rules' utility, which is used for selection and adjusted at the end of each evaluation.

There exists many other models that follows the Pittsburgh approach. In the STEPS model proposed by Kennedy [84], the rule sets are represented by using Escher, a high level strongly typed declarative high level language. Hekanaho proposed the model DOGMA [62], which represents the rules in a generalized first order logic language. Basett and De Jong [15] proposed a model for the evolution of agent behaviors, where the individuals are represented with a variable length binary string defined as the concatenation of fixed length strings representing rules. The main difference with other approaches is that the bits in the conditions are mapped directly to the sensors of an agent and the bits of the actions to the agent actuators.

1.3.6 Specific representations

Different types of non standard representations are used in evolutionary algorithms when the nature of the application suggests a different approach. In this section, mixed integer and

real representations, and Dedekind representations are discussed.

Mixed integer and real representations

Many optimization problems require to deal simultaneously with discrete and continuous variables. In this case, the usual representation for individuals is as a vector that contains both integer an real variables. Formally, an individual I that contains n real variables and m integer variables is defined as the vector $I = (x_1, \ldots, x_n, y_1, \ldots, y_m)$, where $x_i (1 \leq i \leq n)$ are the real variables and $y_j (1 \leq j \leq m)$ are the integer variables. The integer values can be represented just as a single value or as a bit string. In the second case, a variation operator can operate with just a part of their representation.

An example of the use of mixed representations is in the evolution of structure and weights of neural networks as performed by Moriarty et al. [115], where real values encode the weights and integer values are used to specify the hidden units and the pattern connection.

Dedekind representations

Dedekind representations were proposed by Surry and Radcliffe [141] as a kind of formal representation for real parameter optimization problems. The authors based their proposal on the idea that appropriate representations should be derived from the statements of beliefs about the search space. For example, in real variables optimization problems, a typical belief is continuity, which establishes that small changes in the parameters leads to small changes in the objective function. An appropriate representation should be designed by keeping this objective in mind. This continuity belief is called *strong causality* in the context of evolutionary strategies and can be ensured for instance by some *Lipschitz* condition on the fitness function, at least in some neighborhood of the maximum.

Dedekind representations are based on the concept of a *Dedekind cut*. A Dedekind cut is a partition of rational numbers in two non empty sets, such that all numbers in one set are smaller than those in the other set. As an example [141], the following Dedekind cut represents the number $\sqrt{2}$:

$$\sqrt{2} = (\{x \in \mathbb{Q}^+ | x^2 < 2\}, \{x \in \mathbb{Q}^+ | x^2 > 2\})$$

A Dedekind cut represents a single value, and an intersection of Dedekind cuts represents a closed interval in a line. A binary representation for Dedekind cuts can be defined by using $n - 1$ bits to represent n values. In this way, for example, the integer values $0, 1, 2, 3, 4$ can be represented respectively with the binary strings $0000, 0001, 0011, 0111, 1111$. Note that this representation needs more bits than the traditional binary or Gray representations, which need only $\lceil \log_2 n \rceil$ bits. Also, this representation is non orthogonal, meaning that not all combinations of binary values are valid. However, the Dedekind representation presents a locality property: note that when traversing the list of binary values from left to right, when the value of a single bit in a particular position becomes 1, it does not change any more. This locality or continuity property is expected (a belief) to be useful in real parameter optimization problems, something that will be discussed later in sections 1.5.1 and 1.5.2.

Dedekind representations can be easily extended to more than one dimension, by keeping the continuity property on an "axis-by-axis" basis. Iso-Dedekind representations are defined

as an extension of the Dedekind representation, where the concept of continuity applies in all dimensions simultaneously. More details on both representations and comparisons with standard binary coding are presented in [140, 141].

1.4 Selection

The selection operator is used to select well performing solutions from the population [33]. It has a slight different meaning in the different versions of evolutionary algorithms. In genetic algorithms and genetic programming, the selection operator selects good individuals for recombination and mutation, while in evolution strategies and evolutionary programming is used to select good individuals from the result of the application of the other operators. In the so called *steady state* evolutionary algorithms, both types of selections are used [149, 104].

Listing of algorithm 1 (page 11) shows the two types of selections: line 6 corresponds to the so called *parental selection*, which is used to select parents for recombination and mutation, and line 10 corresponds to the *replacement selection*, used to select individuals from the set of children. Note that there is a fundamental difference between the two types of selections: in the parental selection, an individual can be selected several times, while in the replacement selection, an individual is either selected (and survives) or is not selected (and dies).

The selection operators can be compared based on their *selective pressure*, an indicator on the time needed for the best individual to occupy the complete population by its repeated application [33]. Selection operators with a large selective pressure can produce a fast reduction of the genetic diversity in the population, which can be undesirable when it happens too soon producing a convergence to a local minimum, an effect that is called *premature convergence*. On the other side, selection operators with a small selective pressure produce slow convergence, but allow the other operators to perform a better search of the space.

It is common to use a technique know as *elitism*, where the best individuals are incorporated directly to the next instance of the population, in order to avoid loosing them. However, when combined with selection operators with large selective pressure, elitism can produce a premature convergence.

Details on the most used selection operators are presented below.

1.4.1 Deterministic selection

The standard selection methods in evolution strategies are implemented in deterministic terms [132]. For example, in the $(\mu + \lambda)$-ES, the selection operator selects the best μ individuals from the union of μ parents and λ children. In this way, the best solution found in any generation is preserved. In the (μ, λ)-ES, the selection operator selects the μ best solutions from the set of λ children, meaning that there is no guarantee that the best solutions will be preserved during the evolutionary process.

1.4.2 Proportional selection

One of the first proposed selection methods is the *proportional selection* [56, 104, 54] in which the selection probability of a single individual is proportional to its fitness value. A usual

implementation is through the *roulette wheel* algorithm, where sections of the wheel with a width proportional to the fitness are assigned to the individuals. The sampling procedure is performed by spinning the roulette wheel and selecting the individual with the winning section.

This method is highly dependent on the scaling of the fitness function, and in particular it does not work properly when all individuals have a similar fitness or when there are just a few that have a very high fitness value compared with the value of the other individuals in the population [121]. In the first case, the evolutionary algorithm behaves like a random search process, and in the second case, some individuals tend to dominate the selection process and a premature convergence of the population is likely to happen [44]. One solution is the *fitness scaling* [54], a technique that enforces a prescribed *selection pressure*, the ratio between the best and the average fitness. However, fitness scaling does not completely solve the problem of *super-individuals*, and is almost not used any more nowadays.

1.4.3 Rank selection

In the *ranked selection* the individuals are ranked based on their fitness value [57]. The probability of selection of a particular individual is determined based on the rank ordering of the individual in the population, and not directly by its fitness value. In the most simple implementation called *linear ranking*, the selection probability of an individual is linear with respect to its rank order. The rank of an individual, or its position in the population, is defined as 0 for the worst and $\mu-1$ for the best, assuming a population of size μ. Also *nonlinear ranking* can be defined by assigning the selection probability of an individual based on the rank, but not proportionally to it.

Rank based methods do not need fitness scaling and are particularly useful when it is difficult to specify an objective fitness function. Moreover, because it only requires comparisons of individuals, it is insensitive to super-individuals. Moreover, it can be used with any comparison operators, as for instance in the framework of multi-objective optimization [34].

1.4.4 Tournament selection

Other option is the *tournament selection*, where the selection is a two step procedure: a subset of individuals is uniformly selected from the population and the best element of the group is selected as the winner. The size of the group of individuals participating in the tournament is called the *tournament size*. The larger the tournament size, the larger the selective pressure. The get an even lower selection pressure, stochastic tournament can be used: 2 individuals are uniformly selected, as in a deterministic (see above) tournament of size 2, and the best one is selected with probability $p, p \in [0.5, 1]$. The case $p = 1$ is exactly the case of deterministic tournament of size 2, while the case $p = 0.5$ corresponds to no selection at all (uniform selection of the parents).

The method is simple to implement. Moreover, as was the case for the rank selection, only comparisons of fitnesses are required.

1.5 Variation operators

The variation operators apply stochastic transformations to individuals to form new solutions. The most typical variation operators are recombination and mutation, however other kind of operators that cannot be classified in these categories have been defined, like the inver-over operator [104].

The parameters of the operators can be fixed during the evolution or can be updated dynamically [41, 42, 43]. In some cases, variation operators can produce non feasible individuals, introducing the need for repair algorithms [109].

Next sections introduce details on the most usual recombination and mutation operators.

1.5.1 Recombination

The recombination operator, or crossover operator, builds new individuals by exchanging information from individuals of the population known as the parents [23, 54, 104]. The expected behavior of this operator is to get, with greater than zero probability, individuals with a higher fitness by combining good characteristics of the parents. This, of course, will not be always the case, but it is expected that lower fitness individuals produced after a recombination will be discarded later during the evolutionary process.

The frequency of application of the recombination operator is controlled by a parameter (p_c in algorithm 1) known as the *crossover rate*. This value is fixed in some applications of evolutionary algorithms but can change dynamically or can be adapted by the evolutionary process [136].

There exists a number of crossover operators defined in the literature, which of course, depends on the representation used. Next sections provide details on the most used recombination operators, classified by the representation to which they are applied.

Recombination operators for bit strings

The simplest recombination operator is the *one point* crossover. Two individuals are selected from the population for recombination and then two children (or offspring) are created by interchanging genetic material from segments of both parents defined by a single random cut point [105].

Other often used recombination operator is the *multi-point* crossover, or *n-points* crossover, where there are many cut points defined for the interchange [71]. In most cases, the number of points selected for the interchange is 2.

In the *uniform* crossover, the elements at randomly selected positions are interchanged [23, 104]. Other extensions of this operator include the use of different probabilities to select the elements from one parent or from the other, or the use of a mask, which can also evolve during the evolutionary process [11].

Recombination operators for vectors of real numbers

A variety of specific crossover operators for vectors of real numbers have been proposed. However the standard one-point, multi-points and uniform crossover can also be applied [23].

One recombination operator defined specifically for vectors of real numbers is the *arithmetical* crossover proposed by Michalewicz [107], which produces two children from two parents, obtained as the linear combination of two parent vectors. Given two vectors $\overrightarrow{x_1}$ and $\overrightarrow{x_2}$, the resulting children are:

$$\begin{array}{ll} \overrightarrow{x_1}' & = a\overrightarrow{x_1} + (1-a)\overrightarrow{x_2} \\ \overrightarrow{x_2}' & = (1-a)\overrightarrow{x_1} + a\overrightarrow{x_2} \end{array} \tag{1.2}$$

where $a \in [0...1]$.

The *geometrical* crossover [108] searches for a global solution in the boundary of feasible solutions. Given two vectors $\overrightarrow{x} = (x_1, \ldots, x_n)$ and $\overrightarrow{y} = (y_1, \ldots, y_n)$, the resulting child is:

$$\overrightarrow{z} \quad = (\sqrt{x_1 y_1}, \ldots, \sqrt{x_n y_n}) \tag{1.3}$$

Michalewicz defines also the *simple* crossover [107] as a standard one point crossover where care is taken to avoid getting values out of a convex space.

The *heuristic* crossover [107] proposed by Wright produces a single child from two parents using the objective function to determine the direction of the search. Given two vectors $\overrightarrow{x_1}$ and $\overrightarrow{x_2}$, the resulting child $\overrightarrow{x_3}$ is defined as

$$\overrightarrow{x_3} \quad = r(\overrightarrow{x_2} - \overrightarrow{x_1}) + \overrightarrow{x_2} \tag{1.4}$$

where $r \in [0..1]$ and $\overrightarrow{x_2}$ is not worse than $\overrightarrow{x_1}$. Unfeasible children are not accepted.

Deb et al. [35, 34] proposed the simulated binary crossover (SBX) and the parent centric crossover (PCX), which belong to the so called parent-centric recombination operators, where the offspring are created near the parents, assigning to each parent an equal probability of creating an offspring in their neighborhood. PCX operates at vector level, computing the children based on all component values at the same time, while SBX operates at component level, by considering independently each real valued parameter.

Recombination operators for permutations

Standard recombination operators, like one point crossover, cannot be applied to permutations since in most cases, illegal representations of individuals can be obtained. A large number of operators defined specifically for permutations have been proposed in the literature. Just as an example, the order crossover from Davies creates one child from two parents by selecting a segment from one of them, and copying in sequence the elements that are not included in the segment from the other parent. More information on this and other crossover operators can be found in [23, 104, 141].

Recombination operators for parse trees

The standard crossover operator for parse trees interchanges random subtrees from both parents [23]. This crossover definition proposed by Cramer [30] is simple and effective, and it is made possible by the recursive definition of parse trees. In order to simplify the definition of this crossover operation, it is usually assumed that all functions and terminal nodes belong to the same data type, an assumption generally called *closure principle* [90]. Since subtree crossover tends to increase the size of the individuals, usually the size of offspring is limited in number of nodes or depth [23].

Recombination operators for rules

The definition of the crossover operators for rule based individuals depends on the followed approach.

Due to the characteristics of the representation, standard crossover operators are used in the Michigan approach, like one point crossover or multi-point crossover [92, 147].

In the Pittsburgh approach [50], a number of crossover operators have been used. Most implementations use the standard one point crossover and multi-point crossover with the cut points defined in the boundaries between the rules, in order to minimize the disruptive effect. However, these operators cannot introduce new rules, even if they can create individuals with different rule bases. Some implementations, like GABIL [70, 72], allows the cut points to be defined by crossing individual rules. Other standard operators are the *generalization* and *specialization* crossovers used for example in DOGMA [62]. In the generalization crossover, the new child contains the set of antecedents of both parents as the new antecedents of the rules, and in the specialization crossover, the common antecedents of both parents.

Recombination operators for specific representations

Recombination in mixed real and integer representations usually follows the standard recombination operator definitions, like one point or multi point crossover. For example, Moriarty et al. [115] uses just single one point crossover on a population of individuals that represents hidden units, weights and connections of a neural network.

The idea under the recombination operators in Dedekind representations is original. Surry and Radcliffe [140] propose the definition of generalized recombination operators, which are instantiated when applied to specific representations defined based on the beliefs about the search space. In this way, the recombination operators behave like templates used to define the actual recombination operators to be applied during evolution.

For example, the recombination operator R^3 [141] when applied to two individuals defined with the Dedekind representation, produces one offspring represented by the intersection of the Dedekind cuts defined by its parents. Formally, the crossover applied to the parents x and y (represented with single integer values in one dimension) with a parameter $k \in [0, 1)$, produces the child:

$$z = x + k(y - x + 1)$$

The same recombination operator, when applied to binary coded individuals corresponds to the uniform crossover.

1.5.2 Mutation

The mutation operator builds new individuals by applying small random perturbations or changes in the representation. The frequency of application of the mutation operator is controlled by a parameter (p_m in algorithm 1) known as the *mutation rate*. This value is fixed in some applications of evolutionary algorithms but can change dynamically or can be self-adapted by the evolutionary process [11, 27, 135].

There exists a number of mutation operators defined in the literature, which of course, depends on the representation used. Next sections provide details on the most used mutation

operators, classified by the representation to which they are applied.

Mutation operators for bit strings

For binary strings, the usual mutation operator just performs a bit-flipping in one of the positions of the individual [10]. Given the vector $\overrightarrow{x} = (x_1, \ldots, x_n)$, the mutated individual $\overrightarrow{x}'$ is defined as the vector $\overrightarrow{x}' = (x'_1, \ldots, x'_n)$, where:

$$x'_i = \begin{cases} x_i & u > p_{mppb} \\ 1 - x_i & u \leq p_{mppb} \end{cases} \tag{1.5}$$

where u is a random number uniformly distributed in the range [0...1]. This mutation thus amounts to flip each bit independently with probability p_{mppb}, *mutation probability per bit*, not to be confused with the parameter p_m in Algorithm 1, which represents a mutation probability per individual.

Another type of mutation for bitstrings is the *k-bits mutation*, that consists in flipping exactly k uniformly chosen bits of the string (usually, $k = 1$). Though the k-bit mutation and the standard mutation seems to have a similar effect with $P_{mppb} = \frac{k}{n}$ (if n is the length of the bistring), they can have very different behaviors on different problems [52].

Mutation operators for vectors of real numbers

The standard mutation operator for vectors of real numbers consists in the addition of a vector of random values [10]. Usually, given a vector $\overrightarrow{x}$, the mutated individual $\overrightarrow{x}'$ is defined as

$$\overrightarrow{x}' = \overrightarrow{x} + \sigma N(0, C) \tag{1.6}$$

where σ is the *step-size* of the mutation, and C a positive definite matrix, such that $N(0, C)$ is a random vector generated with a multivariate normal distribution with mean zero and covariance matrix C.

It is clear that the result of applying Gaussian mutation will highly depend on the tuning of its parameters (σ and C). The state-of-the art of Gaussian mutation is today the CMA-ES method [60, 59], in which σ and C are adaptive parameters (see section 1.5.3).

However, when adaptive method do not apply (for example in the case of mixed real-integer representations), other mutations or real-valued parameters are available.

The *uniform* mutation proposed by Michalewicz [107] replaces one of the components of the vector by a uniformly drawn value. Given the vector $\overrightarrow{x} = (x_1, \ldots, x_k, \ldots, x_n)$ and a position k selected randomly, the mutated individual is the vector

$$\overrightarrow{x} = (x_1, \ldots, x'_k, \ldots, x_n) \tag{1.7}$$

where x'_k is a random value distributed uniformly in the corresponding range. A modification of this operator, is the *boundary mutation*, where the new value is selected as one of the limit values in the corresponding range.

The *non-uniform* mutation [104, 107] modifies its disruptive effect depending on the stage of the evolutionary process, providing large changes in individuals in the beginning and small changes at the end. Given the vector $\overrightarrow{x} = (x_1, \ldots, x_k, \ldots, x_n)$, the mutated individual is the vector $\overrightarrow{x} = (x_1, \ldots, x'_k, \ldots, x_n)$, where x'_k is defined as follows:

$$x'_k(t) = \begin{cases} x_k(t) + \Delta(t, ub_k - x_k(t)) & if\, u < 0.5 \\ x_k(t) - \Delta(t, x_k(t) - lb_k) & if\, u \geq 0.5 \end{cases} \tag{1.8}$$

where the range of x_k is $[lb_k \dots ub_k]$ and u is a random number uniformly generated in the range $[0, 1]$. $\Delta(t, y)$ is a function defined as follows:

$$\Delta(t, y) = y(1 - r^{(1-\frac{t}{T})^b}) \tag{1.9}$$

where r is a uniform random number in the range $[0, 1]$, T is the maximum generation number and b is the so called shape parameter that determines the dependence on the number of generations. The function Δ returns a value in the range $[0, y]$ with the probability to get a value closer to y that decreases when the number of the current generation approaches to the maximum number of generations. The effect of this mutation operator is to perform big exploratory steps in the beginning of the evolutionary process and small steps at the end.

Mutation operators for permutations

Typical mutation operators for permutations include different variants of the insertion of elements in different positions, swapping elements and the scramble of the complete permutation. The most common operator is the *2-opt*, which given a sequence of elements, reverses the segment between two points selected randomly. Generalization to k points have been also presented. More details can be found in [9, 141].

Mutation operators for parse trees

The most used mutation operators [9] for parse trees are the *grow*, *shrink*, *switch* and *cycle* mutation. The grow mutation consists in the selection of a leaf node and its replacement by a subtree. The shrink mutation consists in the replacement of a random internal node by one of its subtrees. The switch mutation selects an internal node and reorders its arguments. The cycle mutation randomly selects a node and replaces it with a primitive function with the same number of arguments. Other approaches (for example [119]) provide more emphasis in the selection of the subtrees, in order to enhance the quality of the subtrees that are not currently adequate. More information can be found in [4, 9].

Mutation operators for rules

In the original classifier systems from Holland, the mutation is implemented as a standard bit flip operation since the rules are represented in binary form. In the XCS classifier systems from Wilson, the real values that are used to represent the classifiers are modified by adding a random vector [147], in a similar way as the mutation of real valued vectors.

In representations based on the Pittsburgh approach, the same operators can be applied if the representation allows to. For example, in DOGMA [62] the mutation is implemented as a simple bit flipping since representation is a binary string. However, other operators can be applied even by using this simple representation [37]. For example, in GABIL the *adding alternative* operator [73] includes an extra alternative for a variable in the antecedents of a single rule. It is implemented as a normal bit flipping mutation with an asymmetric

probability: the bit is set to 1 with a probability of 0.75 and to 0 with a probability of 0.25. In SAMUEL [116] the specialization mutation reduces in generality the rule antecedents, when a low strength rule produces a highly rewarded behavior.

Mutation operators for specific representations

The simplest approach to perform mutation on individuals represented with mixed integer and real representations, is to consider the integer numbers as real values, and perform standard mutation, like Gaussian random perturbation, with the real values obtained in the integer positions rounded to the nearest integer. However, the effect of rounding can produce non optimum values [10]. Other option is to mutate separately the integer and the real values, however, there can be many interdependencies between them which can affect the performance of the mutation operation. As a solution to these problems, Back [13] suggested the inclusion in the representation of a vector of mutation parameters for the integer fields, which is mutated applying the same rules of the real values.

Surry and Radcliffe [141] propose the definition of generic operators like BMM, which are defined for any specific representation provided that some *minimal* mutation exists for that particular representations.

When instantiated for the Dedekind representation, the mutation is implemented as a bit flip in one of the two possible bits that can be modified to produce an offspring that is a direct neighbor of the original individual. For example, the bit string 0011 can be mutated into 0111 or 0001, with no other option. As a very interesting result, when the number of bits in the representation tends to infinite, the same mutation operator behaves like a Gaussian mutation.

1.5.3 Parameters setting

The effectiveness of an evolutionary algorithm depends mainly on the representation and the definition of the operators. However, the selection of the parameters, like mutation rate, crossover rate, population size, etc, is a key element in the process of defining an evolutionary algorithm. The selection of their values determines the efficiency of the search process and even if the algorithm can find near-optimum solutions or not [41].

The standard approach for parameter setting, usually called *parameter tuning*, consists in choosing *adequate* values selected by previous experimentation. The evolutionary algorithm is then executed with these values, which remain fixed for the complete run. Standard heuristics have been defined and can be used as a guide in parameter setting [135]. However, the selection of adequate values is not easy: the interaction between the different parameters is complex, and the number of experiments to be performed with all combinations of values in order to get significant results is usually too large. Even worse, adequate parameter settings for some experiments might not be adequate in other contexts.

In the other approach, usually called *parameter control*, the values of the parameters are allowed to change during the execution of the evolutionary algorithm. Parameter control methods can be classified in deterministic, adaptive or self-adaptive depending on the update method [41]:

deterministic : the parameters get different values according to a deterministic schedule prescribed by the user. For example, the schedule for a parameter p can be defined as follows:

$$p(t) = 1 - 0.9 * \frac{t}{T}$$

where $p(t)$ corresponds to the value of the parameter p at time t, and T is the maximum number of generations that the evolutionary algorithm will be run. In this example, the initial value of the parameter p is 0.9 and its value will be reduced linearly to reach 0.1 at the end of the evolutionary process.

adaptive : the value of the parameters are updated based on feedback obtained from the search process, rewarding or punishing explicitly the operators according to their effect on the evolution. One example of this method is the update rule of Rechenberg [58] used in evolutionary strategies, where the value of the control parameter is increased if the number of successful mutations over some period of time is larger than $1/5$, and reduced otherwise. Another case of an adaptive parameter is the already-cited CMA-ES method for real-valued representations [60, 59].

self-adaptive : the parameters are encoded into the individuals and their values are modified by the evolutionary process itself. The individuals $I =< x, \sigma >$ consist of the object variable vector x and the parameters σ. The evolutionary algorithm evolves both, usually performing first mutation and/or recombination on the parameters σ, and then applying mutation and/or recombination with the updated parameter values on x.

The most well-known example of successful self-adaptive parameter is the self-adaptive Gaussian mutation [132]. As a simple example, consider the Gaussian mutation where the covariance matrix is set to Id (the identity matrix) in equation 1.6. In this case, the only parameter to tune is σ, the step-size of the mutation. The mutation first modifies the σ parameter using a log-normal mutation:

$$\sigma = \sigma \; e^{N(0,\tau_0)}$$

where τ_0 is a (second-order, and hence rather robust) parameter of the algorithm. The object variables of the individual x are then modified as usual, according to equation 1.6 using the new value of σ.

The evolutionary algorithm hence performs an implicit control of the parameters by evolving their values in the same way as the object variables defined in the representation of the individuals, and though the selection acts on the fitness, that only depends on the object variables. Individuals that do not have the values of the mutation parameters that fit the current region of the search space will be outperformed by those whose mutation parameters have better values.

1.6 Conclusions

Evolutionary algorithms are an optimization method which provides with elegant simplicity a powerful approach to deal successfully with problems that are difficult to consider with other

approaches. But of course, it should not be expected evolutionary algorithms will be superior to any alternative approach in search for *all* optimization problems, as well established by the no-free-lunch theorem (NFL) [150]. However, evolutionary algorithms provide high quality solutions when applied to optimization problems defined by difficult, ill-behaved, objective functions with many local optima, when compared with standard methods like gradient-based techniques when those apply. Even more, evolutionary algorithms can be applied in many problems where other methods cannot be applied at all [129].

One of the advantages of evolutionary algorithms is the fact that only the value of a fitness function is required to perform the evolution, since it is a stochastic zero-th order method. There is no need for gradient information or input output examples. As a consequence, the designer has more freedom in developing solutions. For example, it is typical that the designer is restricted to use differentiable membership functions in fuzzy system design when gradient based methods like back-propagation are used [114]. This restriction is dropped when evolutionary algorithms are used, as will be shown in the model proposed in this thesis.

The parallel search of evolutionary algorithms is one of its main strengths: instead of selecting one solution and refining it as other methods, evolutionary algorithms explore a set of solutions in parallel. In this way, evolutionary algorithms are less likely to get trapped into local minima and the search process is less sensitive to the initial parameter settings. Due to the parallel nature of evolutionary algorithms, a large number of parallel implementations have been proposed in the literature [25, 2].

Evolutionary algorithms are quite robust, even if parameter selection is an important point (see section 1.5.3), wrong parameter settings do not prevent the algorithm to obtain good results [131]. However, in order to obtain near optimum solutions, careful settings of parameters should be performed. The search performed by the evolutionary algorithm should consider the so called *exploration/exploitation dilemma* [39]. Search must be balanced between exploration of the space in the early stages in order to cover the search space as much as possible, and exploitation in later stages in order to enhance the best solutions found by the previous exploration.

However, there are, of course, some disadvantages of evolutionary algorithms. First of all, there is no guarantee that evolutionary algorithms will find the optimum solution, due to the stochastic nature of the method [129]. Also, even if the method requires only a fitness function to perform the optimization, usually the number of evaluations of this function is very high, something that can be prohibitive in some applications. Last, but no least, it should be mentioned that unfortunately, there are currently not so many theoretical results that can justify the application of evolutionary algorithms in some domains, or ensure a high probability of convergence in some problems. However, work is being carried and results are starting to appear, like for example the modeling of evolutionary algorithms as Markov chains [48], studies on operator definitions [52], studies on the interactions of variation operators [16], etc.

Chapter 2

Fuzzy Systems

This chapter introduces the basic concepts on fuzzy systems that will be used in the thesis, together with information on their use in control and a description of the main representative models. Sections 2.1, 2.2 and 2.3 present the basic concepts of fuzzy sets, fuzzy variables and fuzzy inference respectively. Section 2.4 introduces the general model of a fuzzy system, with details on the two main representative instances: the Mamdani and the Takagi Sugeno models. The different types of partitions of the input space are discussed in section 2.5. Sections 2.6 and 2.7 present representative implementations of fuzzy systems and recurrent fuzzy systems applied to control.

2.1 Fuzzy Sets

Fuzzy sets were introduced by Zadeh [156, 157, 158] as a generalization of the concept of a regular set. In a regular set, or crisp set, the membership function assigns the value 1 to the elements that belong to the set and 0 to the elements that do not belong to it. In fuzzy sets, elements can have different degrees of membership to the set.

Formally, a fuzzy set A is characterized by a membership function μ_A that assigns a degree (or grade) of membership to all the elements in the universe of discourse U [101]:

$$\mu_A(x) : U \longmapsto [0, 1] \tag{2.1}$$

The membership value is a real number in the range $[0, 1]$, where 0 denotes definite no membership, 1 denotes definite membership, and intermediate values denote partial membership to the set. In this way, the transition from non membership to membership in a fuzzy set is gradual and not abrupt. Figure 2.1 shows an example of a fuzzy set.

Let us now briefly recall some useful definitions. Let A be a fuzzy set defined in U [21]:

Definition 1 *The* support *of a fuzzy set is defined as the set of elements that have a non-zero membership value (see figure 2.2):*

$$support(A) = \{x \in U, \mu_A(x) > 0\} \tag{2.2}$$

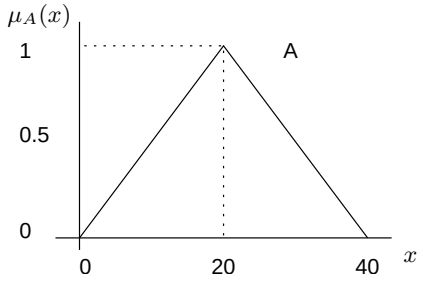


Figure 2.1: An example of a fuzzy set. The fuzzy set A is defined on $\mathbb{R}$. The element 20 has maximum membership ($\mu_A(20) = 1$) and the elements in the intervals $]-\infty, 0]$ and $[40, +\infty[$ minimum membership ($\mu_A(0) = \mu_A(40) = 0$). Elements between 0 and 40 get intermediate membership values.

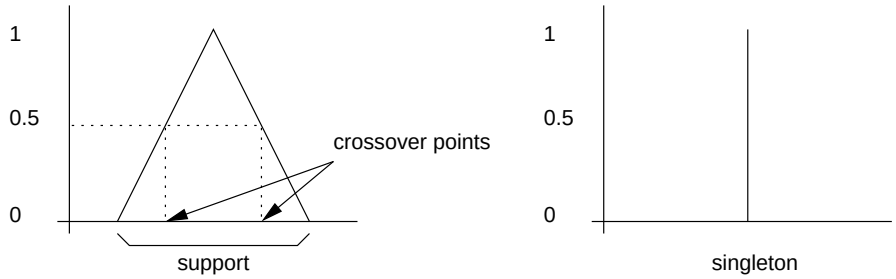


Figure 2.2: Example of the support, the crossover point and the singleton concepts

Definition 2 *A fuzzy set is normal if there is at least a point in its domain with membership one:*

$$(A \text{ is normal}) \Leftrightarrow (\exists x \in U, \mu_A(x) = 1) \tag{2.3}$$

Definition 3 *The elements of the universe with a membership value of 0.5 are called crossover points (see figure 2.2):*

$$(x \in U \text{ is a crossover point for } A) \Leftrightarrow (\mu_A(x) = 0.5) \tag{2.4}$$

Definition 4 *A fuzzy singleton is a normal fuzzy set with a single point support (see figure 2.2):*

$$(A \text{ is a singleton}) \Leftrightarrow (support(A) = \{x\}, \mu_A(x) = 1) \tag{2.5}$$

The operations on fuzzy sets are defined in terms of their membership functions. Let A and B be two fuzzy sets in U, the operations of union, intersection and complement in their most usual form are defined as follows [95](see figure 2.3):

Definition 5 *The union of A and B is defined as:*

$$C = A \cup B \quad \mu_C(x) = max\{\mu_A(x), \mu_B(x)\} \tag{2.6}$$

Definition 6 *The intersection of A and B is defined as:*

$$C = A \cap B \quad \mu_C(x) = min\{\mu_A(x), \mu_B(x)\} \tag{2.7}$$

Definition 7 *The complement of A is defined as:*

$$\overline{A} = U - A \quad \mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{2.8}$$

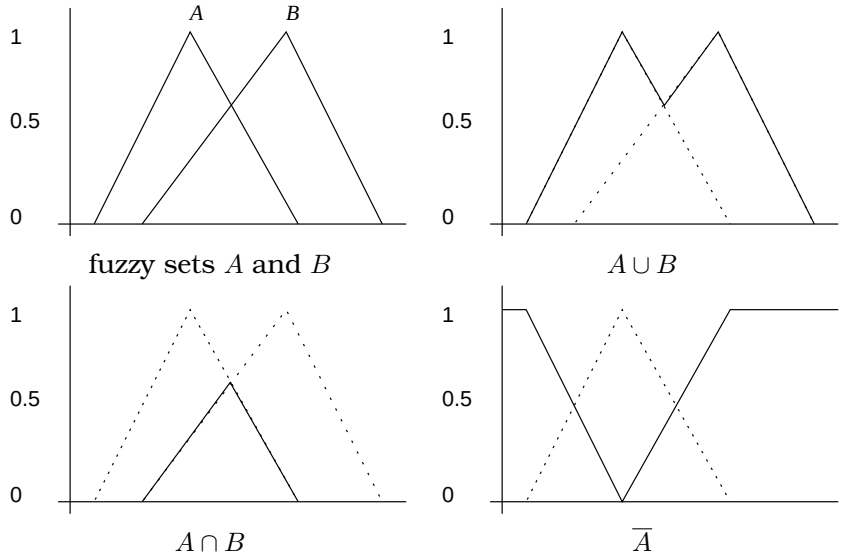


Figure 2.3: Example of the basic operations on fuzzy sets.

2.2 Linguistic variables

A linguistic label is a concept associated to a fuzzy set, defined in terms of its membership function. A linguistic variable is a system variable with its values defined with linguistic labels (or linguistic terms) [158]. As an example, figure 2.4 shows the definition of a linguistic variable with three linguistic terms.

It is desirable, and in most cases implicitly assumed that the linguistic terms associated to a linguistic variable cover the complete input space [95]. Given a linguistic variable V defined in U with terms $A_1, \ldots, A_n$:

Definition 8 *The finite set of fuzzy subsets $\{A_1, \ldots, A_n\}$ of U is a finite fuzzy partition of U if [120]:*

$$\begin{array}{ll} 1. & \sum_{i=1}^{n} \mu_{A_i}(x) = 1 \; \forall x \in U \\ 2. & \textit{every } A_i \textit{ is normal} \end{array} \tag{2.9}$$

Note that the definition of the fuzzy variable *angle* defined in figure 2.4 defines a finite fuzzy partition of the input space, since the summation of the membership to all different terms for a single point in the domain is always 1, and all fuzzy sets are normal. The concept of finite fuzzy partition is important, since it guarantees that every value in the universe U belongs in the same degree to one or more fuzzy sets when compared with other values, or in other words, the semantic level of representation by linguistic terms is the same for every element in U.

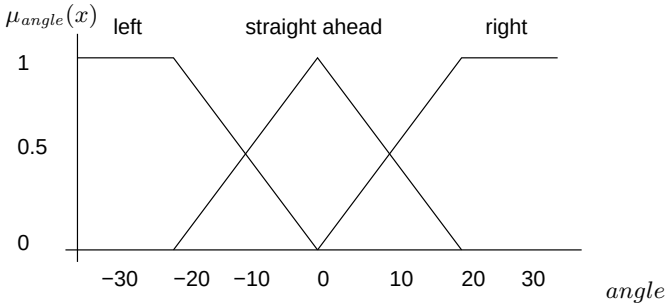


Figure 2.4: An example of a linguistic variable defined with three linguistic terms. The linguistic variable *angle*, which corresponds to the system variable *robot's target angle*, is defined with three linguistic terms: *left*, *straight ahead* and *right*. These linguistic terms define three concepts through three fuzzy sets.

Definition 9 *The partition $part(V)$ defined by V is ϵ-complete if every element in the universe has a membership not smaller than ϵ:*

$$(part(V) \text{ is } \epsilon\text{-complete}) \Leftrightarrow (\forall x \in U \; \exists A_i \; \mu_{A_i}(x) \geq \epsilon) \tag{2.10}$$

The partition shown in figure 2.4 is 0.5-complete since no value in the input domain can get a membership smaller than the crossover points.

Definition 10 *The partition $part(V)$ defined by V is* consistent *if membership 1 to one fuzzy set implies membership 0 to all other fuzzy sets:*

$$(part(V) \text{ is consistent}) \Leftrightarrow ((\forall x \in U \; \mu_{A_i}(x) = 1) \implies (\mu_{A_j}(x) = 0, \forall j \neq i)) \tag{2.11}$$

The partition shown in figure 2.4 is consistent since for all input values that have membership 1 to a fuzzy set, the membership is 0 for all other fuzzy sets.

2.3 Fuzzy inference

Relations between fuzzy variables are represented by fuzzy rules of the form:

$$if \text{ antecedents } then \text{ consequents} \tag{2.12}$$

where generally the antecedents consist of propositions of the form x *is* A, where x is a fuzzy variable and A one of its terms, and the consequents consist of proposition of the form y *is* B, where y is a variable and B one of its terms. It is possible to draw conclusions from a fuzzy rule by applying a rule of inference, a concept that is defined below.

Let $A_1, \ldots, A_n$ be fuzzy sets defined respectively in $U_1, \ldots, U_n$:

Definition 11 *The Cartesian product [95] of $A_1, \ldots, A_n$ is a fuzzy set defined in $U_1 \times \ldots \times U_n$ with membership function:*

$$\mu_{A_1 \times \ldots \times A_n}(x) = min\{\mu_{A_1}(x), \ldots, \mu_{A_n}(x)\} \tag{2.13}$$

A fuzzy relation denotes a degree of presence or absence of a combination of elements belonging to different sets. Formally:

Definition 12 *A n-ary fuzzy relation R between $A_1, \ldots, A_n$ is a fuzzy set defined in $U_1 \times \ldots \times U_n$ with membership function $\mu_R(x_1, \ldots, x_n)$ [95].*

Definition 13 *Given the fuzzy relations R and S defined respectively in $U \times V$ and $V \times W$, the fuzzy max-min composition $R \circ V$ is the fuzzy relation in $U \times W$ defined with the following membership function [95]:*

$$\mu_{R \circ V}(u, w) = sup_{v \in V}(min(\mu_R(u, v), \mu_S(v, w))) \tag{2.14}$$

where sup selects the maximum value (supremum) of a set of elements and min computes the minimum value of the two arguments. The composition is formed by pairs of elements of $U \times W$ with their membership computed as the maximum of the minimum values of the membership of the individual elements when combined with all the elements from V.

The compositional rule of inference [95, 158] is defined as follows:

Definition 14 *If R is a fuzzy relation in $U \times V$ and $\hat{A}$ is a fuzzy set in U, then the fuzzy set $\hat{B}$ in V induced by R is defined as:*

$$\hat{B} = \hat{A} \circ R \tag{2.15}$$

where:

$$\mu_{\hat{B}}(y) = sup_{x \in \hat{A}}(min(\mu_{\hat{A}}(x), \mu_R(x, y))) \tag{2.16}$$

This reasoning strategy can be described in terms of the *generalized modus ponens* [120]:

$$\begin{array}{ll} \textit{Premise 1} & x \text{ is } \hat{A} \\ \textit{Premise 2} & \textit{if } x \text{ is } A \textit{ then } y \textit{ is } B \\ \hline \textit{Conclusion} & y \text{ is } \hat{B} \end{array} \tag{2.17}$$

where the rule (premise 2) corresponds to the fuzzy relation R, defined with a membership that specifies the degree of truth of the implication $A \implies B$.

2.3.1 An example of fuzzy inference

Since the concepts of fuzzy inference and composition of fuzzy relations are central to the operation of fuzzy systems, this section presents an example intended to clarify them. In this example, two fuzzy relations defined in the context of a postal packages transportation company will be introduced, relating the size of the packages with their weight and transportation cost.

The size of a postal package will be described with the linguistic variable *size* defined with the three linguistic labels *small*, *medium* and *large*, referring each one of them to the concept of a package being of small size, medium size and large size respectively. In the same way, the fuzzy variable *weight* will be described with the three linguistic labels *light*, *medium* and *heavy* to represent three different grades of weight. Finally, a third fuzzy variable labeled

cost defined with the linguistic labels *cheap*, *medium* and *expensive*, will be used to represent three different categories of prices.

The relation between the size and the weight of a typical postal package can then be described by the fuzzy relation R_1 defined between the two fuzzy variables *size* and *weight*, as described by the following table:

R_1		weight		
		light	medium	heavy
	small	1	0.3	0.1
size	medium	0.1	1	0.1
	large	0.3	0.5	1

In this table, the values represent the membership grades of the relation between the different linguistic labels of the fuzzy sets *size* and *weight*. As it can be seen, the values used in the definition assume that a small package usually is light, but do not exclude the possibility of it to be of medium weight or even heavy. Same idea was used to define the other values in the fuzzy relation.

The relation between the weight and the cost can also be defined with a fuzzy relation (denoted by R_2), as show in the following table:

R_2		cost		
		cheap	medium	expensive
	light	1	0.1	0
weight	medium	0.2	1	0.2
	heavy	0	0.2	1

The fuzzy max-min composition $R_3 = R_1 \circ R_2$ of these two relations is a new fuzzy relation that relates size with cost:

R_3		cost		
		cheap	medium	expensive
	small	1	0.3	0.2
size	medium	0.2	1	0.2
	large	0.3	0.5	1

The values are computed as defined in equation 2.14. For example, the value of the first element is computed as:

$$
\begin{aligned}
&R_3(small, cheap) = \\
&\quad sup(min(R_1(small, light), R_2(light, cheap)), \\
&\quad min(R_1(small, medium), R_2(medium, cheap)), \\
&\quad min(R_1(small, heavy), R_2(heavy, cheap))) \\
&\quad = sup(min(1, 1), min(0.3, 0.2), min(0.1, 0.3)) = sup(1, 0.2, 0.1) = 1
\end{aligned}
$$

In order to demonstrate fuzzy inference, let us consider a fuzzy set named *package1*, which specifies that a particular package is small:

size	small	medium	large
package1	1	0	0

Note that the size of this package belongs with membership 1 to the linguistic label *small*, representing a crisp concept. The fuzzy set *weight1* induced by the fuzzy set *package1* and the fuzzy relation R_1, can be computed as specified by equation 2.15:

weight	light	medium	heavy
$weight1 = package1 \circ R_1$	1	0.3	0.1

Just as an example, one the values is computed as:

$$
\begin{aligned}
&weight1(medium) = \\
&\quad sup(min(package1(small), R_1(small, medium)), \\
&\quad min(package1(medium), R_1(medium, medium)), \\
&\quad min(package1(large), R_1(large, medium)), \\
&\quad = sup(min(1, 0.3), min(0, 1), min(0, 0.5)) = sup(0.3, 0, 0) = 0.3
\end{aligned}
$$

It is possible to continue this inference process, by computing the fuzzy set *cost1*, induced by the fuzzy set *weight1* and the relation R_2, as shown below:

cost	cheap	medium	expensive
$cost1 = weight1 \circ R_2$	1	0.3	0.2

Of course, it is not necessary to specify the fuzzy sets as representing crisp values. For example, the fuzzy set *package2* describes a package that has membership 0.3 to medium size and 0.7 to large size, representing a rather large package:

size	small	medium	large
package2	0	0.3	0.7

The induced fuzzy sets *weight2* and *cost2* shown below can be obtained by applying the inference process as described before:

weight	light	medium	heavy
$weight2 = package2 \circ R_1$	0.3	0.5	0.7

cost	cheap	medium	expensive
$cost2 = weight2 \circ R_2$	0.3	0.5	0.7

2.4 Fuzzy Systems

A fuzzy system [159] is a computing framework based on the concepts of the theory of fuzzy sets, fuzzy rules and fuzzy inference. Figure 2.5 illustrates its four main components: a knowledge base, a fuzzification interface, an inference engine (or decision making unit) and a defuzzification interface [8].

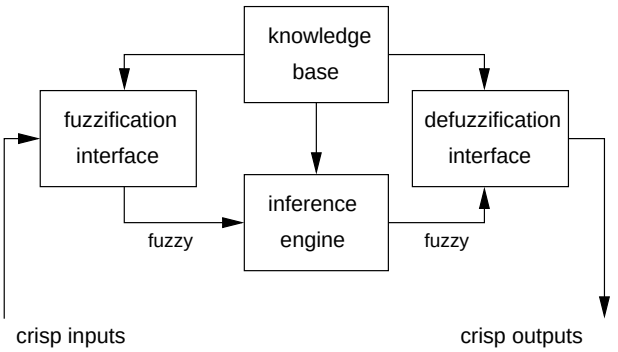


Figure 2.5: The main components of a fuzzy system.

The knowledge base consists of a rule base defined in terms of fuzzy rules, and a data base that contains the definitions of the linguistic terms for each input and output linguistic variable. The fuzzification interface transforms the (crisp) input values into fuzzy values, by computing their membership to all linguistic terms defined in the corresponding input domain. The inference engine performs the fuzzy inference process, by computing the activation degree and the output of each rule. The defuzzification interface computes the (crisp) output values by combining the output of the rules and performing a specific transformation (more details are presented in following sections).

Fuzzy systems are expected to be able to produce an output for every input. This property is called *completeness* [95] and is defined as follows:

Definition 15 *A fuzzy system is complete, or it has the completeness property, if its rule base covers all possible combination of input values.*

Depending on the fuzzy sets used in the definition of the rules in the rule base, the fuzzy systems can be classified as *descriptive* or *approximative* fuzzy systems [63]:

Definition 16 *A fuzzy system is a descriptive fuzzy system if its rule base refers to fuzzy sets externally defined in the database and shared by all the rules.*

Definition 17 *A fuzzy system is an approximative fuzzy system if each rule defines its own fuzzy sets.*

Approximative fuzzy systems can potentially reach more accuracy in the approximation when different fuzzy sets are used for each rule since they provide more degrees of freedom. However, descriptive fuzzy systems have better linguistic interpretation providing higher knowledge comprehensiveness.

Fuzzy systems can be classified in different categories, depending on the shape of the rules and the type of operators used for implementing the modules. The most widely used are the Mamdani and the Takagi-Sugeno models, being also relevant the SAM from Kosko [36, 89, 112] and the Tsukamoto model [8]. All of them can be implemented as approximative or descriptive fuzzy systems. Next sections introduce the characteristics of the main models in terms of a MISO (multiple input single output) structure. There is no loss of generality since a MIMO (multiple input multiple output) fuzzy system can always be decomposed in a group of MISO fuzzy systems [159, 95], as will be shown in section 2.4.3.

2.4.1 The Mamdani Fuzzy System

A standard MISO Mamdani fuzzy system [95, 8, 53] has n input variables $x_1, x_2, \ldots, x_n$ and one output variable v. Each input variable x_i is fuzzified by p_i fuzzy sets A_{ij} ($p_i \geq 1$, $1 \leq i \leq n$, $1 \leq j \leq p_i$) whose membership functions μ_{ik_i} ($k_i = 1, 2, \ldots, p_i$) are arbitrarily defined. The output variable is defuzzified by q fuzzy sets B_i ($q \geq 1$). A complete Mamdani fuzzy system has one rule for each combination of input fuzzy sets. As an example, the definition of the k-th ($1 \leq k \leq \omega$) rule R_k follows:

$$\textit{if } x_1 \textit{ is } A_1^k \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_n^k \textit{ then } v \textit{ is } B^k$$

where the A_i^k are the antecedent fuzzy sets (or antecedent linguistic terms) associated to each input variable x_i, with $A_i^k \in \{A_{i1}, \ldots, A_{ip_i}\}$ and $B^k \in \{B_1, \ldots, B_q\}$.

The linguistic terms for the input variables define regions on the input space where the propositions defined by the consequents apply. Usually the output is computed by the *max-min* inference, which establishes that the activation degree of each rule is computed by applying the *min* operator to the antecedents, and the operator *max* to aggregate the outputs to get the final output value. Figure 2.6 shows an example of the application of the *max-min* inference. Formally, the degree of activation w_k of the rule R_k is computed as the minimum membership value of all antecedent linguistic terms:

$$w_k = min\, \mu_{A_i^k}(x) \tag{2.18}$$

The membership function B' of the output fuzzy set is computed as:

$$\mu_{B'}(y) = max_k[w_k(x), \mu_{B^k}(y)] \tag{2.19}$$

The crisp output value is obtained by applying a defuzzification method. The two most used methods are the mean of maximum (MOM) and the center of gravity (COG) [96]. The method MOM computes the average of the values with the highest membership degree. The method COG computes the center of gravity of the complete fuzzy set. Figure 2.7 shows an example of the computation of the crisp output by using the MOM and the COG methods.

The selection of the defuzzification method plays an important role in the implementation of a Mamdani fuzzy system [153]. Experimentally, it has been found that the defuzzification algorithm tends to dominate the computation time of the whole fuzzy system [53].

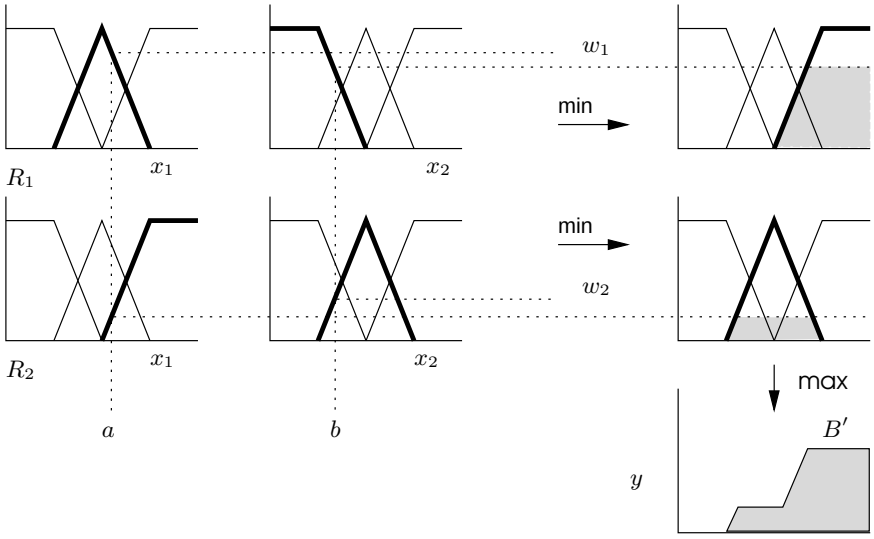


Figure 2.6: The *max-min* inference method as usually applied in Mamdani fuzzy systems. The figure shows a fuzzy system with two input variables and one output variable, all of them defined with three linguistic terms. The rule base contains two rules R_1 and R_2, which involve in their definition the antecedent linguistic terms defined with a wide continuous line. The dotted lines show the membership values for the input values a and b. The minimum values are denoted by w_1 and w_2. The computed fuzzy set B' is shown in gray color.

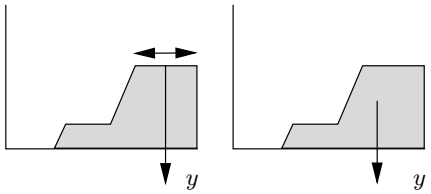


Figure 2.7: An example of the output produced by the mean of maximum (left) and the center of gravity (right) methods when applied to the fuzzy set obtained as the output of the inference process in the example of figure 2.6.

2.4.2 The Takagi-Sugeno Fuzzy System

A MISO Takagi-Sugeno fuzzy system [95, 8, 53, 142] has n input variables $x_1, x_2, \ldots, x_n$ and one output variable v. Each input variable x_i is fuzzified by p_i fuzzy sets A_{ij} ($p_i \geq 1$, $1 \leq i \leq n$, $1 \leq j \leq p_i$) whose membership functions μ_{ik_i} ($k_i = 1, 2, \ldots, p_i$) are arbitrarily defined. The output variable is defined in terms of a linear combination of the input variables. A complete Takagi-Sugeno fuzzy system has one rule for each combination of input fuzzy sets. As an example, the definition of the k-th ($1 \leq k \leq \omega$) rule R_k follows:

$$\textit{if } x_1 \textit{ is } A_1^k \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_n^k \textit{ then } v_k = a_{0k} + a_{1k}x_1 + \ldots + a_{nk}x_n \tag{2.20}$$

where the A_i^k are the antecedent fuzzy sets (or antecedent linguistic terms) associated to each input variable x_i, with $A_i^k \in \{A_{i1}, \ldots, A_{ip_i}\}$, v_k is the value of the output variable v defined by the k-th rule, and a_{0k} and a_{ik} are adjustable real valued parameters. The figure 2.8 shows an example of the evaluation of a Takagi-Sugeno fuzzy system.

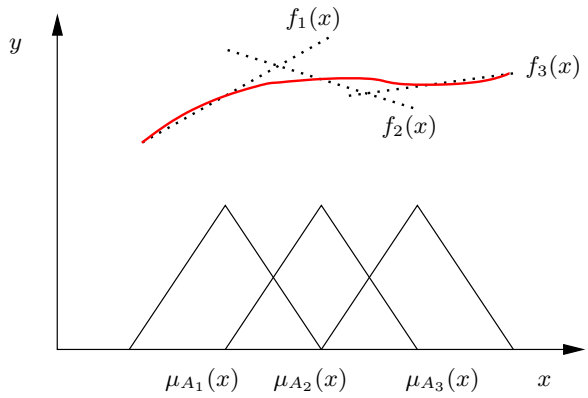


Figure 2.8: The inference method as usually applied in Takagi-Sugeno fuzzy systems. The figure shows a fuzzy system with one input variable x and one output variable y. The rule base contains three rules R_1, R_2 and R_3 which involve in their definition the linguistic terms A_1, A_2 and A_3 and the linear functions f_1, f_2 and f_3 respectively. The output of the fuzzy system corresponds to the value defined by the linear combination of the consequent functions, weighted by their firing strength.

The first step in the evaluation of the fuzzy system is the computation of the membership values $\mu_{A_i}(x_i)$ for each input variable x_i. The membership values are combined into a scalar value that corresponds to the rule firing strength $w_k(x)$ with a standard `t-norm` operator [120], with product and minimum as usual selections. As an example, the product combination is defined as follows:

$$w_k(x) = \mu_{A_1^k}(x_1) \times \ldots \times \mu_{A_n^k}(x_n) \tag{2.21}$$

where $x = (x_1, \ldots, x_n)$ is the complete input vector.

The firing strength of each rule is computed as the ratio of its firing strength to the sum of all rules' firing strengths as follows:

$$\overline{w}_k = \frac{w_k}{\sum_{i=1}^{n} w_i} \tag{2.22}$$

The output of the fuzzy system is then computed by adding the output value produced by each rule multiplied by its weighted firing strength:

$$v(x) = \sum_{k=1}^{\omega} \overline{w}_k * v_k(x) \tag{2.23}$$

The Takagi-Sugeno fuzzy system has less linguistic power when compared with a Mamdani fuzzy system, since the consequents are not represented with meaningful linguistic terms. However, Takagi-Sugeno fuzzy systems are better approximators [154, 155, 8], being a good choice when approximation accuracy is desirable.

2.4.3 MIMO vs. MISO fuzzy systems

As mentioned before, there is no loss of generality by considering only MISO fuzzy systems, since all MIMO fuzzy systems can always be decomposed into a set of MISO fuzzy systems [159]. For example, let us consider a standard MIMO Mamdani fuzzy system [95] with n input variables $x_1, x_2, \ldots, x_n$ and m output variables $v_1, v_2, \ldots, v_m$. Let us assume that each input variable x_i is fuzzified by p_i fuzzy sets A_{ik} ($p_i \geq 1$, $1 \leq i \leq n$, $1 \leq k \leq p_i$) and each output variable is fuzzified by q_j fuzzy sets B_{jk} ($q_j \geq 1$, $1 \leq j \leq m$, $1 \leq k \leq q_j$). The r-th rule of such fuzzy system looks like:

$$\textit{if } x_1 \textit{ is } A_1^r \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_n^r \textit{ then } v_1 \textit{ is } B_1^r \textit{ and } \ldots \textit{ and } v_m \textit{ is } B_m^r$$

where the second subscripts of the fuzzy sets are omitted for clarity reasons. Note that the consequent of the rule specifies a set of m completely independent outputs, associated to each one of the m output variables. Based on this analysis, this single MIMO fuzzy rule is equivalent to a set of m MISO fuzzy rules, where all of them have the same antecedents, but specify different outputs for each one of the output variables, as shown below:

$$\textit{if } x_1 \textit{ is } A_1^r \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_1^r \textit{ then } v_1 \textit{ is } B_1^r$$

$$\textit{if } x_1 \textit{ is } A_1^r \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_1^r \textit{ then } v_2 \textit{ is } B_2^r$$

$$\ldots$$

$$\textit{if } x_1 \textit{ is } A_1^r \textit{ and } \ldots \textit{ and } x_n \textit{ is } A_1^r \textit{ then } v_m \textit{ is } B_m^r$$

As a general rule, a MIMO fuzzy system can be considered as a set of MISO fuzzy systems. The same reasoning can be applied to Takagi-Sugeno fuzzy systems.

2.5 Partition of the Input Space

The cardinality of a complete rule base set R is defined by:

$$|R| = \prod_{i=1}^{n} p_i \tag{2.24}$$

where n is the number of input variables and p_i is the number of linguistic values associated to the i-th input variable. The cardinality of a complete rule base set grows exponentially with the number of input variables. This is a key problem in the development of fuzzy systems, since the computation time depends directly on the number of rules.

There exists a number of techniques used to reduce the cardinality of the set of rules by providing a different partition of the input space. In the standard approach, the domain is divided in a set of boxes (or hyper-boxes) that are parallel to the axes, as shown in the example of figure 2.9a. It is also possible to use the conjunction and negation operators in the definition of the antecedents of the rule in order to partition the input space in a smaller number of regions, as shown in figure 2.9b. It is also possible to define multivariate membership functions, allowing the definition of regions with arbitrary shapes, as shown in figure 2.9c. By considering the number of regions, this last method is the most adequate, since

the number of regions can be significantly reduced for a complete fuzzy system. However, the linguistic interpretation is also reduced, even if a projection of these regions into the individual axes can still be obtained [8].

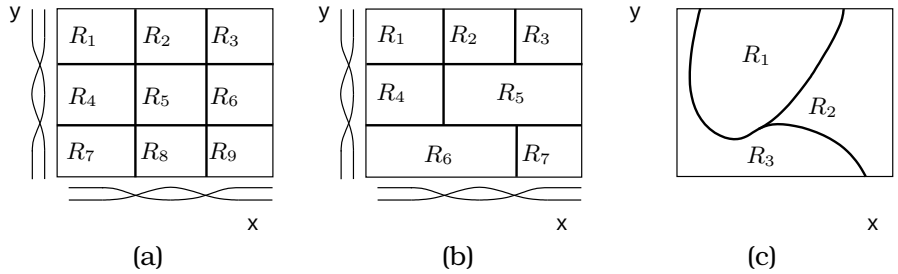


Figure 2.9: Examples of partitions of a two dimensional input space. The partition (a) corresponds to the standard partition in boxes with one region for each combination of linguistic terms. The partition (b) corresponds to a partition where the antecedents are combined with other logic operators like *not* and *or* besides the standard *and*. In partition (c), each region is defined with a multivariate membership function.

Other possibility to reduce the complexity of the partition and the number of rules is to decompose the fuzzy system into a set of interconnected fuzzy systems with smaller number of rules each one [8]. Figure 2.10 shows an example of two fuzzy systems interconnected in such a way that the output of the first system is one of the inputs of the second fuzzy system. A fuzzy system with three input variables with three fuzzy sets each one needs 27 rules; the hierarchic fuzzy system interconnected as shown in figure 2.10 needs just 18.

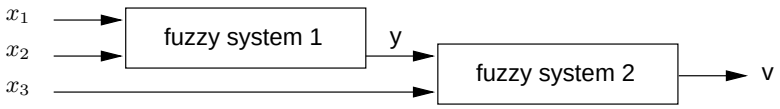


Figure 2.10: Example of a hierarchic fuzzy system with three inputs, one output and two sets of two dimensional rules. The variables x_1 and x_2 are the inputs of the first fuzzy system. The second fuzzy system receives as inputs the variable x_3 and the output of the first fuzzy system.

2.6 Fuzzy Controllers

One of the most successful areas of application of fuzzy systems is control, where fuzzy controllers have proved to be very effective in the context of controlling complex ill defined processes [1, 45, 95]. A fuzzy controller (FC) [95, 120] is a standard fuzzy system defined to control a target process, usually called a *plant* in control terminology. Figure 2.11 shows the execution model of a fuzzy controller in a closed-loop control system: the actual control signals are generated by the fuzzy controller based on the status of the process.

The fuzzy controller is usually designed by formulating the knowledge of a human expert into a set of linguistic variables and fuzzy rules [63]. However, this process is difficult and

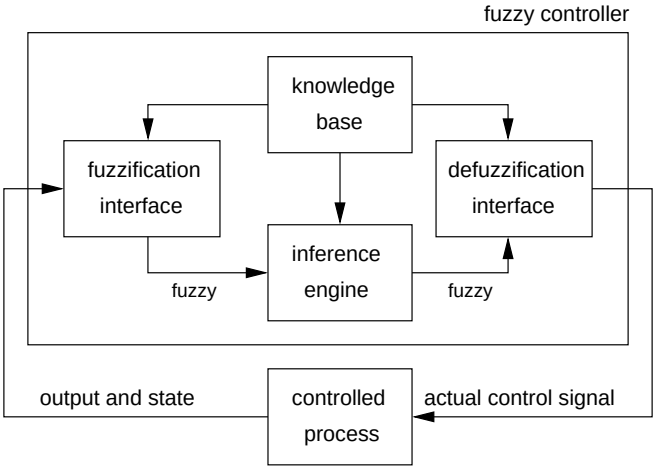


Figure 2.11: The execution model of a fuzzy controller.

tedious due to the lack of knowledge or understanding of the process to be controlled [1, 134]. To make the problem even more difficult, there is no formal framework for the design of fuzzy controllers [100].

A large number of methods to automate the building process and to evaluate and fine tune the obtained fuzzy controllers have been proposed in the literature, with methods based on analytical techniques, reinforcement learning, neural networks and evolutionary algorithms being the most successful ones [103, 126, 38].

Some of these methods are not based in human expert knowledge represented in terms of fuzzy concepts. For example, in the OLS method proposed by Wang [145], a fuzzy system with the same number of fuzzy basis functions as the number of input output examples is defined as an initial approach, and then the most significant basis functions are selected. In the synthesis method proposed by Galichet and Foulloy [51], a fuzzy system equivalent to a given PI (proportional integral) linear controller is built automatically. The input membership functions are distributed regularly in the input space and a set of generic rules is defined in such a way that chosen input output points (modal values) belong to the control surface. Output values for non modal inputs are correctly generated by linear interpolation in a Takagi-Sugeno fuzzy system or with a careful selection of the defuzzification operators in a Mamdani fuzzy system.

Data driven methods can be used to enhance the quality of fuzzy systems. Typically, an initial model structure for a fuzzy controller is created from expert knowledge, which is translated into a collection of fuzzy rules. The parameters in the structure (membership functions, weights of the rules, etc.) are then fine tuned using input output data by applying analytical methods like least squares techniques [144]. An example of these techniques is the method proposed by Takagi-Sugeno [142] which finds the least squares estimates of the consequent parameters with a heuristic process, enlarging the fuzzy structure till no improvement is observed.

Neural network based algorithms can also be used to automate the design based on numerical data as well as on expert knowledge since most fuzzy control structures can be directly

mapped into feed-forward neural network models. The combined approach provides advantages from both worlds: the low level learning and computational power of neural networks is joined together with the high level human-like thinking and reasoning of fuzzy systems [99]. This combination has been very successful and there is a large number of models that combines fuzzy systems with neural networks [99], or even with standard PID control [64, 139]. The simplest method to combine neural networks with fuzzy systems is to train neural networks with data generated from a fuzzy controller [87]. The obtained neural network in some cases can provide a smoother behavior and superior robustness [100]. The most usual approaches involve storing the fuzzy information in the structure and weights of neural networks in order to obtain a smaller, more efficient controller [88] or to fine-tune the parameters of the fuzzy controller [20, 68].

Evolutionary algorithms have also been used to design or tune the different components of fuzzy controllers [29, 63]. A fuzzy control system is represented as a set of rules by following the Michigan or Pittsburgh approach. In the simplest approach, the set of parameters of the fuzzy system defines the optimization space and the evolutionary algorithm operates by searching the optimal values of the parameters. One of the first approaches using genetic algorithms is the proposal by Karr [77], where each individual encodes with a binary string the fuzzy partitions of all input and output variables defined in terms of trapezoidal membership functions. The rule base is defined by using expert knowledge and it is not updated by the evolutionary algorithm. A more ambitious approach performs a structure and parameter search in order to define the rule base and the parameters at the same time [28, 76, 97, 122]. One of the first proposals that performs both structure and parameter learning is the proposal from Homaifar et al [65], where each individual represents a complete rule set and the partition of the input variables by using an integer based codification. Some approaches base the representation on neural network models, using the learning capabilities of the neural networks in order to enhance the search abilities of the evolutionary algorithms [3, 124, 133].

Next sections present details on some approaches followed in the development of fuzzy controllers. They are not the latest or the best ones, but they have been selected because they are conceptually simple and provide the basis for most other models, including the model introduced in this thesis.

2.6.1 The ANFIS model

The Adaptive Network based Fuzzy Inference System (ANFIS) model was proposed by Jang in 1993 [68] as a basis for constructing a set of fuzzy rules with appropriate membership functions from a set of input output examples. This model has been a source of inspiration for many other fuzzy models defined in more recent works.

The ANFIS model is an adaptive network defined with five layers, as shown in figure 2.12. Each node on the first layer has associated a linguistic term and its function is to compute the membership of the input values to the corresponding linguistic term. Nodes in the second layer (labeled as $\prod$) represent fuzzy rules. They compute the firing strength w_i of the rules, by evaluating the product of the membership of the input values to the fuzzy terms that correspond to the antecedents of the rule. Nodes in the third layer (labeled as N) compute the normalized firing strength $\overline{w}_i$ by dividing each rule firing strength by the summation of all of them. Nodes in the fourth layer compute the weighted output of the rules by evaluating the

Takagi-Sugeno type linear approximator f_i multiplied by the normalized firing strength. The output of the system is computed by the node in the last layer as a summation of all incoming signals.

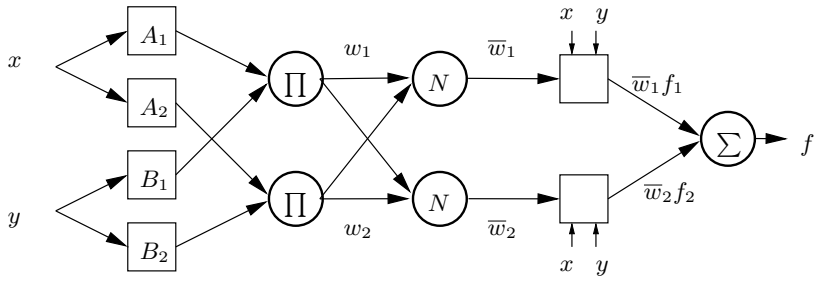


Figure 2.12: An ANFIS network defined with two inputs (x and y), one output (f) with two linguistic terms defined for each input variable (A_1, A_2 and B_1, B_2 respectively) and two fuzzy rules.

The ANFIS model has parameters in the nodes of the first and the fourth layers, which correspond respectively to the parameters of the fuzzy terms of the antecedents and the parameters of the linear approximators in the consequents of the rules. A hybrid learning algorithm based on the standard back propagation algorithm is proposed to fine tune the values of the parameters. Successful results of experiments on control and non linear function approximation problems are presented by the authors [68].

2.6.2 The GARIC model

In 1992 Berenji and Khedkar [20] proposed the Generalized Approximate Reasoning based Intelligent Control (GARIC) model as an architecture that learns and tunes a fuzzy controller by using a single binary failure indication. The architecture has three main components (see figure 2.13): an Action Selection Network (ASN) that maps the status vector into a recommended action F using fuzzy inference, an Action Evaluation Network (AEN) that maps the status vector and the failure signal into a scalar score v used to produce the internal reinforcement $\hat{r}$, and a Stochastic Action Modifier (SAM) that produces the output to be applied to the plant based on the recommended action F and the internal reinforcement.

The ASN is the fuzzy controller. Its structure is very similar to the five layers structure of the ANFIS model [68]. Learning occurs by fine tuning the free parameters of both the ASN and the AEN, which in the case of the ASN correspond to the shape of the linguistic terms. Learning is performed by modifying the parameters in such a way that the scalar score v of the current action is maximized. During learning, the AEN acts like a critic element that predicts the reinforcement associated with the current situation. The SAM selects the current action to be applied to the plant by randomly perturbing the suggested action depending on the current reinforcement signal: perturbation is large when the last action is considered as incorrect (low reinforcement $\hat{r}$) and small, following the recommendation of the ASN, when the last action is good (high reinforcement $\hat{r}$).

This model was extended to include rule generation ability by using evolutionary learning [79, 80].

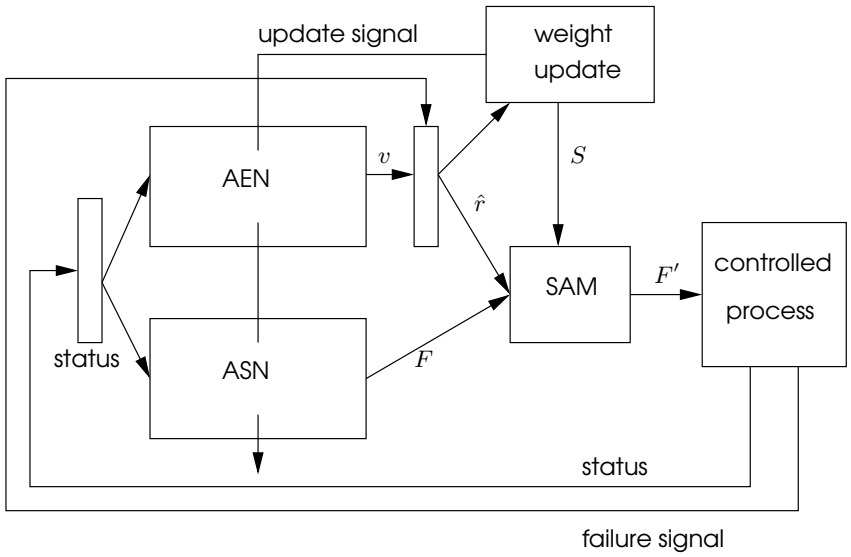


Figure 2.13: The architecture of the GARIC model.

2.6.3 The genetic learning model of Hoffmann

Hoffmann proposes two simple, but representative genetic learning models for fuzzy controllers [63]: in the first model the individuals encode the partition of input and output variables and in the second, individuals encode fuzzy control rules.

Input variable partitions are encoded by a vector of real numbers that correspond to the center of triangular fuzzy membership functions (see figure 2.14). There is no need to encode the width of the membership functions since it is assumed that normalized membership functions that sum up to 1 are used. The output fuzzy membership functions are encoded by a vector of real numbers that correspond to the positive offsets between adjacent fuzzy sets. This simplification is allowed since the defuzzification algorithm proposed depends only on the center of the output fuzzy sets and not on the wide or the overlap between them.

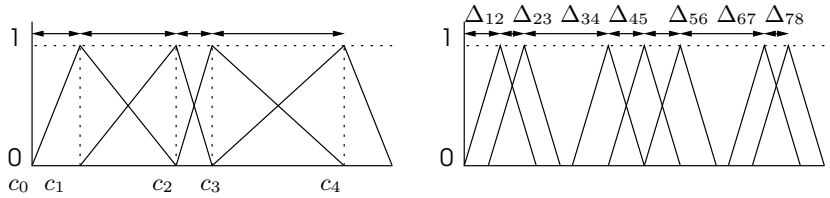


Figure 2.14: The partition of input (left) and output (right) fuzzy variables in the model proposed by Hoffmann. Only the centers c_i of the input fuzzy membership functions and the distance Δ_{ij} between adjacent output fuzzy membership functions need to be encoded.

Rules are encoded by a fixed length vector that represents the complete relation between input and output variables in terms of a complete grid partition. Each position corresponds to a particular entry in the rule base table and contains the index of the output fuzzy set

associated to that entry. The size of the individuals grows rapidly with the number of variables and fuzzy sets. For example, a system with two input variables partitioned each one with five fuzzy sets and one output variable needs a vector of 25 integer values ($5 * 5 = 25$).

The representation can be easily modified to implement a Takagi Sugeno fuzzy system by replacing the entries in the rule base by a set of real values that correspond to the parameters of the linear approximator defined by each rule.

2.6.4 The SEFC model

The Symbiotic Evolution based Fuzzy Controller (SEFC) model was proposed by Juang in 2000 [76] as a design method for fuzzy controllers that uses ideas from symbolic evolution [115]. Each individual encodes a single rule by specifying the centers and the widths of bell shaped membership functions. A fuzzy system is built for evaluation by randomly selecting a set of individuals (see figure 2.15). The process is repeated a number of times in order to guarantee that all rules have been sufficiently selected for conforming fuzzy systems. The fitness of an individual (a rule) is computed as the summation of the fitness of all fuzzy systems in which it participate with other individuals divided by the number of combinations.

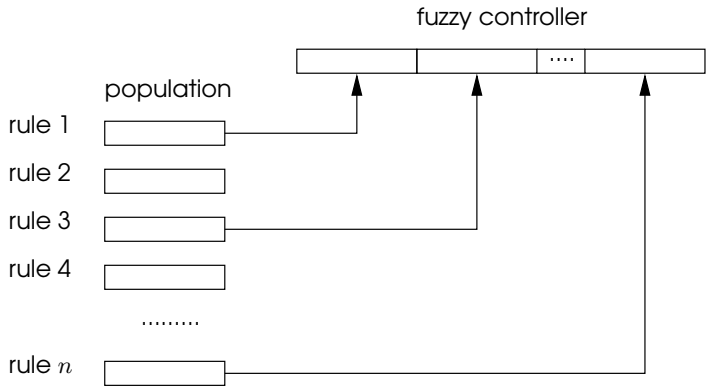


Figure 2.15: The SEFC model. A fuzzy system is built for evaluation by randomly selecting a set of rules from the population.

An individual represents a partial solution to the problem since complete solutions needs several individuals. Unlike the standard evolution where the population converges to an optimal (or suboptimal) individual, in a symbiotic evolution algorithm a single solution cannot dominate the population since it needs from other individuals in order to provide a solution. Experiments show that high quality and fast results are obtained with this approach [76].

2.7 Recurrent fuzzy controllers

The domain of application of standard fuzzy systems is limited to static problems due to its feed forward structure [94]. Most non linear problems in control require the processing of temporal sequences, or in other words, in these problems the output depends on the current

input and previous values of inputs and/or outputs. A very interesting approach that considers small order temporal problems with fuzzy logic is the FTR model proposed in [26, 118]. However, unless the number of delayed inputs and outputs is known before, it is not possible to define a feed forward model without backward connections that can process temporal sequences [75]. This is the case in most control problems, where this information is usually not known. On the other hand, recurrent structures can deal with this kind of problems. A large number of neural network models have been proposed which are essentially feed forward structures with an extra set of memory units used to store previous activation values that are connected back to the inputs of other units.

By considering the amount of recurrent neural network models that have been proposed, it is expected to see that most recurrent fuzzy systems are based on neural networks. For example, the RFNN model (Recurrent Fuzzy Neural Network) proposed in [94] defines recurrent connections in the second layer of the structure, which correspond to the units that encode the membership antecedent values. The RSONFIN (Recurrent Self Organizing Neural Fuzzy Inference Network) model proposed in [74] perform structure and parameter learning and includes an extra layer of units with recurrent connections that provides a kind of internal memory. The DFNN (Dynamic Fuzzy Neural Network) model proposed in [102] includes recurrent neural networks in the consequent in place of standard linear approximators like in the TS model. The TRFN (Takagi-Sugeno Type Recurrent Fuzzy Network) model proposed in [75] has an extra unit with recurrent connections for each fuzzy rule, which is responsible of memorizing the temporal history of activations of the rule. Other models like [78], [46] and [160] follow similar approaches. In most cases, both supervised learning and non gradient based algorithms, like evolutionary algorithms or reinforcement learning, have been used to build or enhance the models.

Next sections presents details on the above mentioned approaches.

2.7.1 Fuzzy temporal rules

The FTR model proposed by Cariñena et al. in [26, 118] and on her PhD thesis, allows to define fuzzy temporal rules that are used to establish relations of dependency between occurrences of events. In the antecedents of the rules it is possible to use temporal relations like "just after" and "just before" to refer the value of a variable respectively in the next and in the previous time step. Relations like "a few instants before" can be used to represent the values of the variables a definite number of time steps in the past. An example of a rule defined by following the FTR proposal follows:

if	x_1	*raises sharply a few instants before and*
	x_2	*falls sharply*
then		*there is a door on the left wall*

In this example (simplified from [26]) there are two input variables and one output variable. The terms "raises sharply" and "falls sharply" correspond to temporal relations that are defined by the difference between the values of the variables at the current time and at some definite time in the past. For example, by considering two time steps, the events "raises

sharply" and "falls sharply" can be defined respectively as follows:

$$x(t) - x(t-2) > PV \tag{2.25}$$

$$x(t) - x(t-2) < -PV \tag{2.26}$$

where PV is a positive threshold value, and $x(t)$ is the value of the input variable x at time t.

The TFR model is a simple but very interesting approach with no recurrent connections, where the number of previous inputs is fixed when the controller is defined. The temporal fuzzy rules representation has been used for landmark detection in mobile robotics [26], for detecting doors by using a Nomad 200 robot, avoidance of moving objects [118] and wall following.

2.7.2 The RFNN model

The RFNN model proposed by Lee et al. [94] is a neural network based fuzzy system that includes recurrent connections in the second layer of the structure. The structure is shown in figure 2.16.

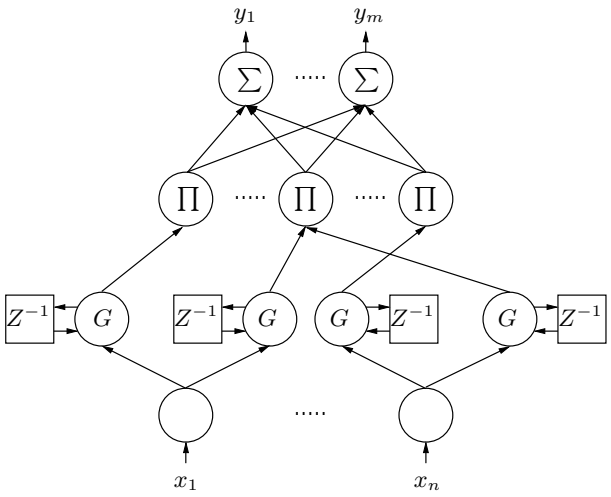


Figure 2.16: The structure of the RFNN model.

It is a typical four layer structure, with input nodes in the first layer that performs no computation, and are just used to distribute the values, nodes that compute a Gaussian membership in the second layer (labeled as G), nodes that represent fuzzy rules in the third layer (labeled as $\prod$) and nodes that compute the outputs in the fourth layer (labeled as $\sum$).

Each node in the second layer has associated an extra node labeled as Z^{-1} which acts as a feedback node, providing the memory capacity that allows the model to develop temporal representation abilities. The output produced by the nodes labeled as G in the second layer depends on the value computed as membership in the current time step and the values computed in previous time steps.

Learning is performed by applying the back propagation algorithm. There is an extra weight in the backward connections that is also optimized during learning. The model was

successfully applied to time sequence prediction, identification of nonlinear dynamic systems and identification of a chaotic system [94].

2.7.3 The RSONFIN model

The RSONFIN model proposed in [74] is a self organizing fuzzy neural network that includes context elements that act like internal memory organized in a feedback layer. The structure of the RSONFIN model is shown in figure 2.17.

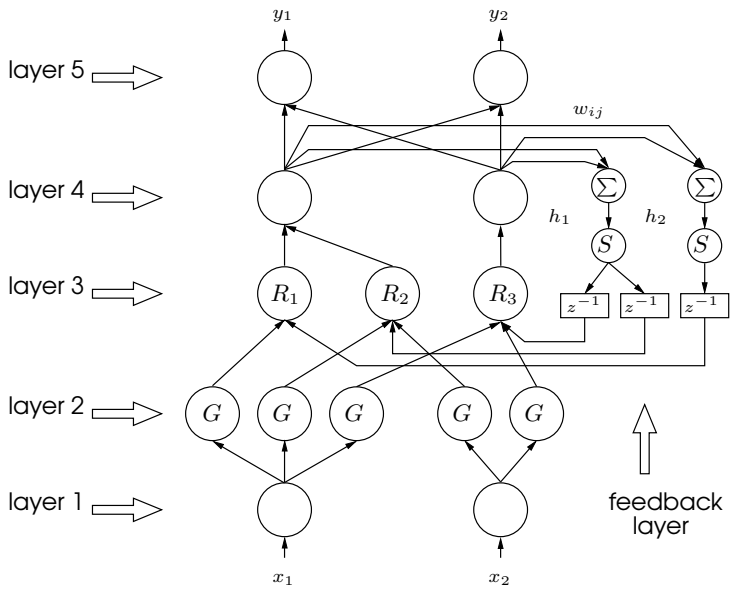


Figure 2.17: The structure of the RSONFIN model.

The nodes in the first layer are input nodes that perform no computation, passing the input values to the next layer. Units in the second layer (labeled as G) compute the membership of the input values by using Gaussian membership functions. The units on the third layer correspond to the rules of the fuzzy system and compute the firing strength of each rule by considering the current membership provided by the nodes of the second layer combined with the temporal firing degree provided by the recurrent units. The units in layer four perform LMOM defuzzification [20] by using a multidimensional Gaussian fuzzy set and the nodes in layer five combine the outputs of the rules. The feedback layer contains context nodes (labeled as $\sum$) associated each one with a feedback term node (labeled as S) in a number that is equal to the number of nodes in layer four. The context nodes act as defuzzifiers of internal hidden rules providing the value of the internal variables h_i combining the output of the rules with the weights w, which represent fuzzy singleton numbers [8]. The feedback nodes compute the firing history of the fuzzy rules by applying a sigmoid membership function. These values are used in the next time step for the computation of the strength of fuzzy rules. The nodes labeled as S, which performs the sigmoid computation, acts like a kind of global fuzzy set, and it was adopted to simplify the network structure [74].

The i-th rule of a RSONFIN model with n inputs, 1 output and m internal variables is defined as follows:

$$\begin{array}{ll} \textit{if} & x_1(t) \textit{ is } A_{i1} \textit{ and } \ldots \textit{ and } x_n(t) \textit{ is } A_{in} \textit{ and} \\ & h_i(t) \textit{ is } S \\ \textit{then} & y_1(t+1) \textit{ is } B_{i1} \textit{ and} \\ & h_1(t) \textit{ is } w_{1i} \textit{ and } \ldots \textit{ and } h_m(t) \textit{ is } w_{mi} \end{array}$$

where $x_i(1 \leq i \leq n)$ are the input variables, y_1 is the output variable, $h_j(1 \leq j \leq m)$ are the internal variables, $w_{ji}(1 \leq j \leq m)$ are the weights associated to the connections.

The training is performed incrementally. In a first stage, the number of rules and the partition of the input space is determined with fuzzy clustering. In the second stage, the parameters are tuned by using a recursive learning algorithm based on gradient information. The model was successfully applied to time sequence prediction, adaptive noise cancellation, identification of nonlinear dynamic systems and plant control [74].

2.7.4 The DFNN model

The DFNN model proposed in [102] is a fuzzy model, where the consequents of the rules are implemented with recurrent neural networks. Each neural network becomes a kind of local approximator in the area of application defined by the antecedents of the corresponding rule. The i-th rule is defined as follows:

$$\begin{array}{ll} \textit{if} & x(t) \textit{ is } A^{(l)} \\ \textit{then} & \hat{y}_l(t) = RNN_l(x(t)) \end{array}$$

where $x(t)$ is the input vector at time t, $A^{(l)}$ is a fuzzy region in the premise part and RNN_l is a neural network that implements the consequent part of the rule.

The general representation of the model is shown in figure 2.18. The premise and the defuzzification parts are static, while the consequents implemented by the recurrent neural networks define a dynamic system due to their feedback connections.

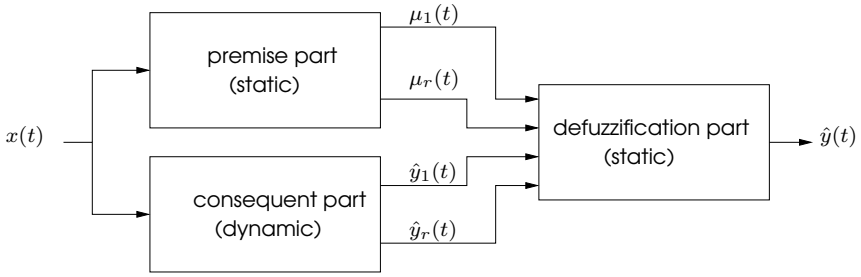


Figure 2.18: The general representation of the DFNN model.

Learning is performed with an algorithm that updates the parameters of the neural network based on the concept of constrained optimization. The model was successfully applied to system identification and noise cancellation problems [102].

2.7.5 The TRFN model

The Recurrent Fuzzy Network (TRFN) model proposed in [75] is a Takagi Sugeno fuzzy system that has an extra unit with recurrent connections for each fuzzy rule, which is responsible of memorizing the temporal history of activations of the rule.

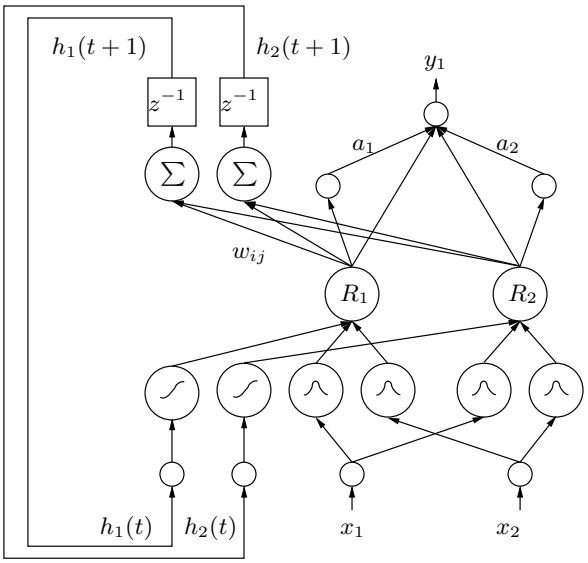


Figure 2.19: The general representation of the TRFN model.

Figure 2.19 shows the structure of a TRFN defined with two inputs x_1 and x_2, a single output y_1 and two rules R_1 and R_2. The two rules can be defined formally as:

$$\begin{array}{lll} & \textit{if} & x_1(t) \textit{ is } A_{11} \textit{ and } x_2(t) \textit{ is } A_{12} \textit{ and } h_1(t) \textit{ is } G \\ \text{Rule 1:} & \textit{then} & y(t+1) = a_{10} + a_{11}x_1(t) + a_{12}x_2(t) + a_{13}h_1(t) \textit{ and} \\ & & h_1(t+1) \textit{ is } w_{11} \textit{ and} h_2(t+1) \textit{ is } w_{21} \end{array}$$

$$\begin{array}{lll} & \textit{if} & x_1(t) \textit{ is } A_{21} \textit{ and } x_2(t) \textit{ is } A_{22} \textit{ and } h_2(t) \textit{ is } G \\ \text{Rule 2:} & \textit{then} & y(t+1) = a_{20} + a_{21}x_1(t) + a_{22}x_2(t) + a_{23}h_1(t) \textit{ and} \\ & & h_1(t+1) \textit{ is } w_{12} \textit{ and} h_2(t+1) \textit{ is } w_{22} \end{array}$$

where A_{ij} and G are fuzzy sets, and w_{ij} and a_{ij} are consequent parameters for the outputs h and y.

The output y is computed by using a standard Takagi Sugeno linear approximator. There is one internal output h for each fuzzy rule. The context nodes indicated by a summation operator compute a defuzzification for each rule and produce an output that is used as an extra input in the next time step. They are used to determine the influence degree of the temporal history of the rule.

The TRFN model can be trained with a supervised learning algorithm when input output data is available or with an evolutionary algorithm in other case. The supervised learning algorithm is a two phase algorithm that builds the structure and then performs the parameter

tuning. Rules are created every time a clustering algorithm determines that the current set of rules does not cover a new input pattern. Parameters are updated with a gradient based algorithm. In the evolutionary algorithm based approach, each individual contains all the parameters of the TRFN model. Fitness is computed based on the performance of the model evaluated on the problem. The model was successfully applied to dynamic system identification and dynamic control problems [75].

2.8 Conclusions

Fuzzy systems are a very effective tool to solve problems in a variety of domains, being particularly successful in the field of process control. Their main advantage is the possibility to use expert knowledge about a problem when it is available in the form of if-then rules. In this case, the definition of the fuzzy control system is straightforward due to the linguistic nature of the fuzzy variables and the fuzzy rule set. The main disadvantage is that there is currently no standard method to determine the exact membership functions and the most adequate rules in order to maximize the performance of the fuzzy system. As a consequence, this approach produces usually a sub-optimal fuzzy control system. However, this control system can be enhanced by adjusting its parameters. Many approaches have been followed to consider this problem, particularly based on neural networks and evolutionary algorithms.

Neural networks based algorithms are well adapted when training data is available. However, solutions obtained from the learning process usually cannot be interpreted directly, and the neural network behaves like a black box. A combination of fuzzy systems and neural networks can get the advantages of both worlds, providing a system that is an interpretable model capable of learning and capable of using problem-specific *a priori* knowledge. One of the disadvantages is that usually some restrictions are imposed into the fuzzy system, like differentiable requirements for membership functions, in order to allow gradient descent algorithms to work.

Evolutionary algorithms provide the possibility to enhance or determine all components of a fuzzy system by just requiring a fitness measure for each individual. In this way, requirements like input output examples or differentiability of membership functions are not necessary.

Chapter 3

The Voronoi Approximation

This chapter presents the Voronoi approximation, a method that provides data approximation based on computational geometry concepts. The method is a mesh-less approach that has a simple and appealing structure. The approximation is built in terms of two dual concepts: the Voronoi diagram and the Delaunay triangulation. The chapter also introduces the basic concepts and techniques used to define the Voronoi approximations.

The chapter outline is as follows. Section 3.1 presents basic concepts on three useful structures of computational geometry: Voronoi diagrams, Delaunay triangulations and barycentric coordinates. Section 3.2 introduces the Voronoi approximation. Section 3.3 gives details on the evolutionary approach that is proposed to evolve Voronoi approximators together with a parametric study of the evolutionary process. Finally, section 3.4 presents experimental results on two function approximation problems.

3.1 Basic Computational Geometry Concepts

3.1.1 Voronoi Diagrams

A Voronoi diagram is a fundamental and useful geometric structure that induces a subdivision of the space based on a set of points called *sites*. Formally, a Voronoi diagram [31] of a set of N sites $P = \{p_1, \ldots, p_N\}$ in $\mathbb{R}^d$ is the subdivision of the space into N cells $(\mathcal{V}(p_1), \mathcal{V}(p_2), \ldots, \mathcal{V}(p_N))$, one for each site in P, such that a point q lies in cell $\mathcal{V}(p_i)$ if and only if the distance between q and p_i is smaller than the distance between q and p_j for any $p_j \in P$ with $j \neq i$. More formally:

$$q \in \mathcal{V}(p_i) \Longleftrightarrow \textit{dist}(q, p_i) < \textit{dist}(q, p_j)\ \forall p_i \in P, i \neq j$$

where $\textit{dist}(x, y)$ corresponds to the Euclidean distance between the points x and y.

The Voronoi diagram of P is denoted by $Vor(P)$ and the Voronoi cell that corresponds to the site p_i is denoted by $\mathcal{V}(p_i)$. Formally, the Voronoi region defined by the site p_i is defined as follows:

$$\mathcal{V}(p_i) = \{q \mid dist(q, p_i) \leq dist(q, p_j)\ \forall p_j \in P, i \neq j\} \tag{3.1}$$

The vertices of the diagram are called *Voronoi vertices* and the segments *Voronoi edges*. Figure 3.1 illustrates an example of a Voronoi diagram in $\mathbb{R}^2$.

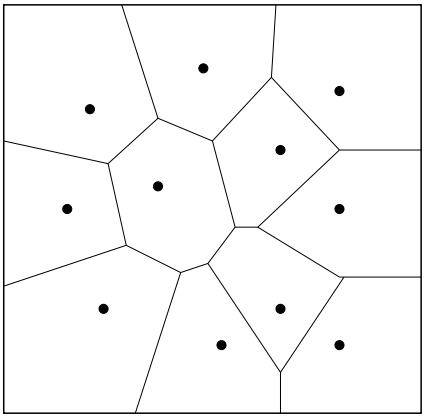


Figure 3.1: An example of a Voronoi diagram for a set of points P in $\mathbb{R}^2$.

A single Voronoi cell $\mathcal{V}(p_i)$ can also be defined as the intersection of the $N-1$ half planes $h(p_i, p_j)$:

$$\mathcal{V}(p_i) = \bigcap_{1 \leq j \leq N,\ j \neq i} h(p_i, p_j) \tag{3.2}$$

where $h(p_i, p_j)$ corresponds to the half plane that contains p_i defined by the perpendicular bisector of the segment $\overline{p_i p_j}$. Figure 3.2 shows an example of a Voronoi cell in $\mathbb{R}^2$ defined in terms of half plane intersections.

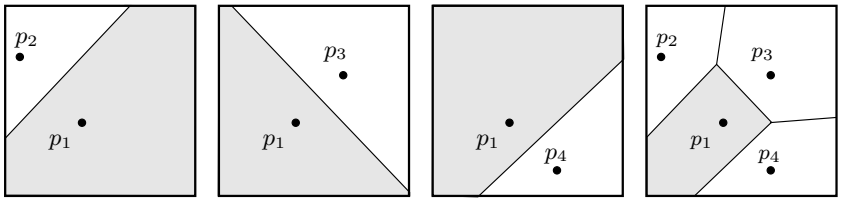


Figure 3.2: An example of a Voronoi cell defined in terms of half plane intersections for a set of 4 sites $P = \{p_1, p_2, p_3, p_4\}$. The first three diagrams correspond to the half planes $h(p_1, p_2)$,$h(p_1, p_3)$ and $h(p_1, p_4)$ that contain p_1 respectively. The fourth diagram shows the full Voronoi diagram, including the cell $V(p_1)$ defined as half planes intersections.

Various algorithms to compute the Voronoi diagram of a set of points in the Euclidean space have been proposed. The optimal algorithms have $O(n \log n)$ time complexity. The interested reader can find a detailed explanation of the most representative algorithms in [31].

3.1.2 Delaunay Triangulations

The dual concept of the Voronoi diagram is the so called Delaunay triangulation. A triangulation S of a set of points P in $\mathbb{R}^2$ is defined as the maximal planar subdivision whose vertex set is P [31]. A maximal planar subdivision S is a subdivision such that no edge connecting two vertices can be added to S without destroying its planarity. Or in other words [31], any

edge that is not in S intersects one of the existing edges. Figure 3.3 illustrates an example of a Delaunay triangulation in $\mathbb{R}^2$.

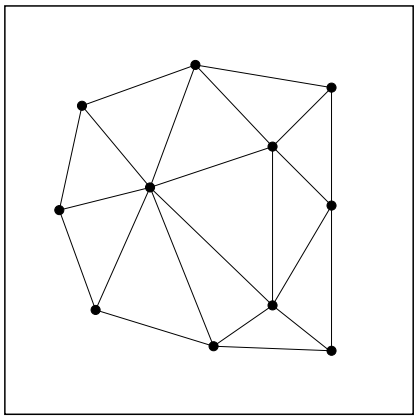


Figure 3.3: An example of a Delaunay triangulation. The set of points that defines the triangulation is the same set P used in figure 3.1.

A maximal triangulation S of a set of points P is a Delaunay triangulation if and only if the circumcircle of any triangle in $\mathcal{S}$ does not contain a point of P in its interior (A circumcircle of a triangle is defined as the circle that goes through its three summits). The Delaunay triangulation is unique for a given set of points, except for some rare degenerate situations. Figure 3.4 shows the circumcircles that define a Delaunay triangulation. These definitions can be straightforward extended to $\mathbb{R}^n$, with $n > 2$.

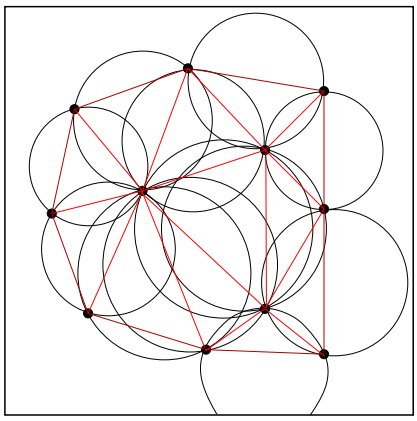


Figure 3.4: The circumcircles defined by the Delaunay triangulation exemplified in figure 3.3. Note that the circumcircles defined by each three groups of points do not include other points, except the three points that define them

The Delaunay triangulation is the dual of the Voronoi diagram. The duality implies that there is a Delaunay edge between two nodes if and only if their Voronoi cells share a common edge. The Delaunay triangulation can be easily derived from a Voronoi diagram, with the

consequence that algorithms used to compute Voronoi diagrams can be used to compute Delaunay triangulations. Various algorithms that follow a different approach have also been proposed, with the randomized incremental approach being one of the most used. See [31] for a detailed explanation.

3.1.3 Barycentric Coordinates

The barycentric coordinates were introduced by Moebius in 1827 as an approach to define local coordinate systems. Given a set of N points $\{p_1, \ldots, p_N\}$ which define a convex set in the plane, the N barycentric coordinates $(\alpha_1, \ldots, \alpha_N)$ of a point p are the coefficients of the following equation:

$$\alpha_1 p_1 + \ldots + \alpha_N p_N = p \tag{3.3}$$

where:

$$\alpha_1 + \ldots + \alpha_N = 1 \tag{3.4}$$

Barycentric coordinates can be interpreted as the relative area of the convex subset defined by the point p and $N-1$ vertices of the original convex set. The barycentric coordinates always sum up to one, cannot be negative and can reach zero if the point p lies in one of the sides of the convex subset.

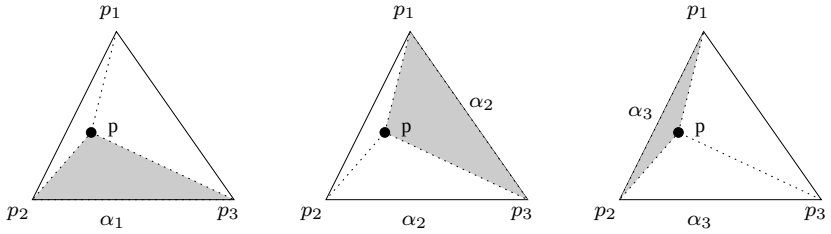


Figure 3.5: An example of the computation of the barycentric coordinates for a given point p in a triangle defined by the points p_1, p_2 and p_3. The relative area of the gray triangle corresponds to the value of the barycentric coordinate.

As an example, figure 3.5 shows in $\mathbb{R}$ the barycentric coordinates $(\alpha_1, \alpha_2, \alpha_3)$ of a point p in a triangle defined by the points p_1, p_2 and p_3. Next definition shows how the value of the barycentric coordinates can be calculated by considering the areas of the triangles involved.

Definition 18 *Given a triangle T in $\mathbb{R}^2$ defined by the points p_1, p_2 and p_3 and a point p included in T, the barycentric coordinates $(\alpha_1, \alpha_2, \alpha_3)$ of p are defined as follows:*

- $\alpha_1(p) = A(p, p_2, p_3)\ / A(p_1, p_2, p_3)$
- $\alpha_2(p) = A(p_1, p, p_3)\ / A(p_1, p_2, p_3)$
- $\alpha_3(p) = A(p_1, p_2, p)\ / A(p_1, p_2, p_3)$

where $A(x_1, x_2, x_3)$ is the area of the triangle defined by the points x_1, x_2 and x_3.

3.2 The Voronoi approximation

The Voronoi approximation [82] is a local approximation method which partitions the input space in Voronoi regions, and apply linear local approximators in each region. The value of the approximation in the center of each Voronoi region is the value defined by the corresponding local approximator. The value in other points of the domain is the linear combination of the values defined by neighbor local approximators in a way that is defined below.

Definition 19 *Given a set of N points $P = \{p_1, \ldots, p_N\}$ in the domain $X \subset \mathbb{R}^n$, the* Voronoi approximation *F for a vector $x = (x_1, \ldots, x_n) \in X$ is defined as:*

$$F(x) = \sum_{i=1}^{N} \phi_i(x) f_i(x - p_i; a_i) \tag{3.5}$$

where the ϕ_i $(1 \leq i \leq N)$ are the shape functions and $f_i(x; a_i)$ are linear functions with parameters $a_i = (a_{i0}, \ldots, a_{in})$, defined as:

$$f_i(x; a_i) = a_{i0} + a_{i1} \cdot x_1 + \ldots + a_{in} \cdot x_n \qquad 1 \leq i \leq N \tag{3.6}$$

The shape functions are defined in terms of the Delaunay triangulation induced by the points in P and define the way in which the local approximators are combined. Formally, they are defined as follows:

Definition 20 *The* shape function *ϕ_k of the Voronoi approximation associated to the site p_k is defined as:*

$$\phi_k(x) = \begin{cases} \alpha_k(x) & x \in \mathcal{T}(p_k) \\ 0 & elsewhere \end{cases} \tag{3.7}$$

where $\mathcal{T}(p_k)$ is a simplex defined by the union of the Delaunay triangles that have no null intersection with the Voronoi region with center p_k, and $\alpha_k(x)$ is the barycentric coordinate associated to p_k in the sub-simplex $\tau \subset \mathcal{T}$ defined by the Delaunay triangulation to which the point x belongs. The simplex $\mathcal{T}$ is defined as follows:

$$\mathcal{T}(p_k) = \bigcup_{\tau \in Del(P),\ \tau \cap \mathcal{V}(p_k) \neq \emptyset} \tau \tag{3.8}$$

where $Del(P)$ corresponds to the Delaunay partition defined by P.

The shape functions associated to the Voronoi regions in $\mathbb{R}^d$ correspond to hyper-triangles of dimension d.

Definition 21 *The support of the shape function ϕ_k corresponds directly to $\mathcal{T}(p_k)$:*

$$support(\phi_k) = \mathcal{T}(p_k) \tag{3.9}$$

As an example, figure 3.6 shows the shape functions associated to a set of four Voronoi regions in $\mathbb{R}$ defined by the points $p_1 = 1$, $p_2 = 3$, $p_3 = 7$ and $p_4 = 9$. The points that define the Voronoi regions (sites) are drawn over the *x*-axis, the vertical lines define the boundaries of the Voronoi regions, and the top arrows indicate the span of each region. The shape functions

are shown as triangles that take the value 1 at the center of the corresponding Voronoi region, 0.5 in the boundaries and 0 in the center of the neighbor Voronoi regions. An exceptions is constituted by the outbound regions, which keep the value 1 from the center point in the outside direction.

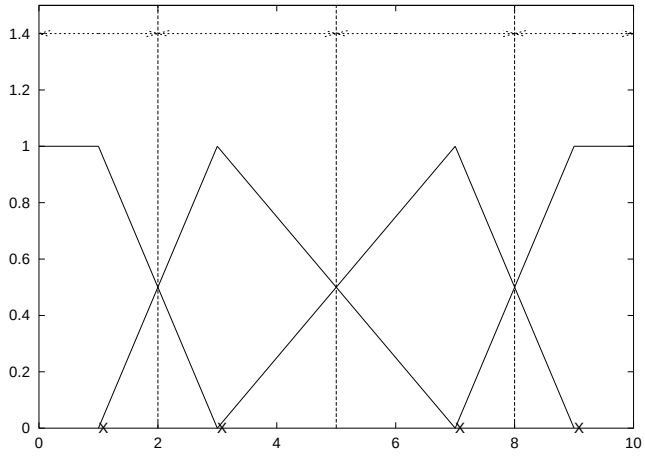


Figure 3.6: An example of the triangular shape functions defined for a set of $N = 4$ points in $\mathbb{R}$. The points are indicated with a cross over the *x*-axis, the vertical lines define the boundaries of the Voronoi regions and the top arrows indicate the span of each one.

Figure 3.7 shows an example of the shape function associated to a Voronoi region in $\mathbb{R}^2$. Figure shows in dotted lines the Voronoi region associated to the site p_3. At the same level in the space, the continuous lines define the six Delaunay triangles that have p_3 as one of their vertices. The shape function ϕ_3 corresponds to the pyramid shaped function in the third dimension. Note that it takes the value 1 in the center of the region and its values go down linearly to reach 0 on the centers of the neighbor Voronoi regions.

The value of the shape function ϕ_k on a given point x depends on the barycentric coordinate of x in the Delaunay triangle τ to which x belongs. Figure 3.7 also shows an example in $\mathbb{R}^2$ of the computation of the barycentric coordinate associated to the point p_3 for a given x. The outer triangle that contains x corresponds to the simplex defined by the Delaunay triangulation to which x belongs. The coordinate value corresponds to the area of the shadowed triangle. Note that the value of the relative area is 1 when x is equal to p_3 and it goes down linearly to 0 on the side of the triangle opposite to p_3. The support of the shape function in figure 3.7 corresponds to the union of all Delaunay triangles defined in the plane $z = 0$.

Note that open Voronoi regions can introduce difficulties in the computation of the Voronoi approximation. In order to avoid the problem of dealing with them, all approximations are computed with extra points that define an hypertriangle of dimension d selected in such a way that the input space X is completely included into it. Each one of these points has associated the local approximator defined as the constant function $f(x) = 0$, producing a drop to 0 in the value of the approximation in the outer Voronoi regions that are inside the input space X. In order to reduce the effect of this fixed approximation at the borders of X, the hypertriangle is defined with a very large size compared with the size of X. As an example,

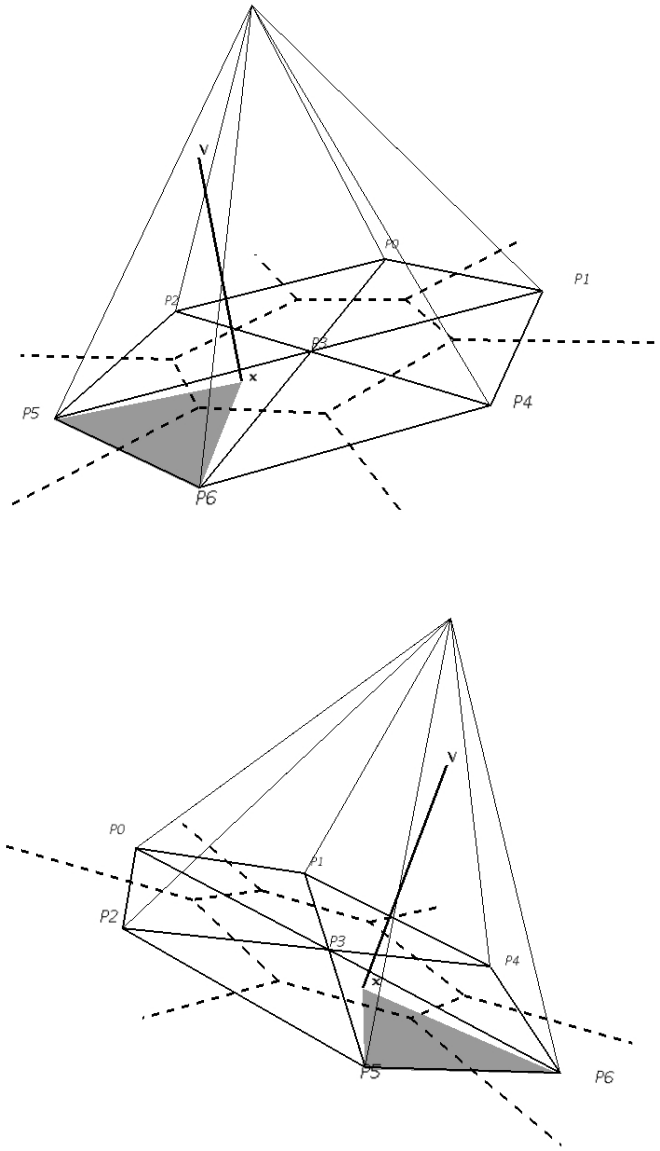


Figure 3.7: An example of the computation of a 2-dimensions Voronoi shape function presented in two different views. The dotted lines represent the Voronoi diagram defined for the points $P_0, \ldots, P_6$. The continuous lines in the same plane define the Delaunay triangulation. The shape function $\phi_3(x)$ is represented graphically by the altitude of the pyramid. The colored triangle shows the computation of the value $V = \phi_3(x)$ for a given x, represented as the relative area of the shadowed triangle.

figure 3.8 shows hypertriangles that includes the input space in $\mathbb{R}^2$ and $\mathbb{R}^3$.

Figure 3.8: An example of the external hypertriangles used to avoid open Voronoi regions in 2D (left) and 3D (right). The dark gray area identifies the input space

In order to simplify the presentation, most diagrams and examples in these chapters will just consider the definition of the Voronoi approximator as defined in equation 3.5 without showing explicitely the extra points that define the external hypertriangle.

Discussion

The Voronoi approximation method is a mesh-less approach that has many elements in common with other local approximation methods. Its definition was particularly influenced by the well known finite element method (FEM) [113, 67], widely used in engineering applications [161]. In the FEM, the domain is discretized into smaller regions called elements, which are connected at specific points called nodes. The complete approximation is a piecewise approximation based in the nodal values.

As a first step to build the approximation with the FEM, the input domain is partitioned by using a mesh pattern, defining the elements and their nodes. In the Voronoi approximation, the partition of the input space is defined indirectly by the selection of a set of points, that will define the partition through its Delaunay triangulation. Many geometric based approaches (like Delaunay triangulations [138]) have also been used in the context of the FEM method.

The shape functions in FEM are usually selected based on the desired properties of the elements, being usual to use linear or quadratic functions, which ensure C^0 continuity between the elements. Note that the standard shape functions in FEM produce value 1 in one of the nodes, and value 0 in the other nodes of the same elements [7], which is exactly the same approach used in the shape functions of the Voronoi based approximation. The same definition of shape functions were used in other Delaunay based partition strategies like the fuzzy sensors partition [17, 19, 18], defined in the context of fuzzy sensors.

Once the shape functions have been selected in FEM, a set of equations is defined, assembling the elements to form a piecewise approximation, which can be solved applying the so called boundary conditions. These conditions prescribe the values of the nodes to some known values, allowing the system to have a single solution, which can be obtained with standard numeric techniques. In the proposed approximation, the partition of the input space and the values of the local approximators are obtained by using evolutionary algorithms, as it will be shown in section 3.3.

3.2.1 Examples

This section provides two examples on the construction of a Voronoi based approximation. Based on a set of points and their associated local approximators, the complete approximations will be constructed graphically.

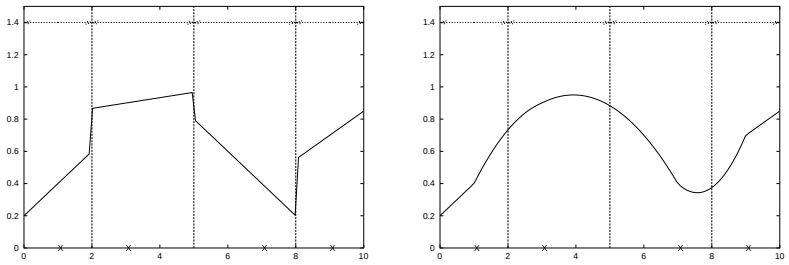
Voronoi approximation in $\mathbb{R}$

As an example, table 3.1 defines a set P of $N = 4$ points in $\mathbb{R}$, with their associated local approximators. The set of points P is the same set used in the example shown in section 3.2. Figure 3.6 shown in that section displays the Voronoi diagram and the shape functions.

point	coordinates	local approximator
p_1	(1)	$f_1(x_1) = 0.39 + 0.19x_1$
p_2	(3)	$f_2(x_1) = 0.89 + 0.05x_1$
p_3	(7)	$f_3(x_1) = -0.19 + 0.42x_1$
p_4	(9)	$f_4(x_1) = 0.71 + 0.12x_1$

Table 3.1: Set of points and local approximators used in the example in $\mathbb{R}$ (Figure 3.6).

Plot *a* in figure 3.9 shows the linear approximators associated to each Voronoi region and plot *b* shows the full Voronoi approximation.



(a) (b)

Figure 3.9: The approximation defined by the local approximators in each Voronoi region (plot a) and the complete Voronoi approximation (plot b) for the example in $\mathbb{R}(d = 1)$.

Note that the value of the Voronoi approximation at the center of each Voronoi region (just over the cross signs at the *x*-axis) is exactly the value of the associated linear approximation, since the associated shape function takes the value 1 while all others take the value 0. As an example, the value of the Voronoi approximation at point p_2 is $f_2(0)$, since $\phi_2(p_2) = 1$ and $\phi_i(p_2) = 0$, for all $i \neq 2$, as shown below:

$$\begin{aligned} F(p_2) &= \phi_1(p_2) \cdot f_1(p_2 - p_1) + \phi_2(p_2) \cdot f_2(p_2 - p_2) + \\ &\quad \phi_3(p_2) \cdot f_3(p_2 - p_3) + \phi_4(p_2) \cdot f_4(p_2 - p_4) \\ &= \phi_2(p_2) \cdot f_2(p_2 - p_2) \\ &= 1 \cdot f_2(0) = f_2(0) = 0.89 \end{aligned}$$

As other example, the value of the Voronoi approximation at the boundaries between the Voronoi regions is the combination in the same degree of the values defined by the corresponding local approximators, since the involved shape functions take the value 0.5 at these points. As an example, the value of the Voronoi approximation at the boundary $b = (p_1 + p_2)/2 = 2$ defined by the Voronoi regions $\mathcal{V}(p_1)$ and $\mathcal{V}(p_2)$ can be computed as follows:

$$\begin{aligned} F(b) &= \phi_1(b) \cdot f_1(b - p_1) + \phi_2(b) \cdot f_2(b - p_2) + \\ &\quad \phi_3(b) \cdot f_3(b - p_3) + \phi_4(b) \cdot f_4(b - p_4) \\ &= 0.5 \cdot f_1(b - p_1) + 0.5 \cdot f_2(b - p_2) \\ &= 0.5 \cdot f_1(1) + 0.5 \cdot f_2(-1) = 0.5 \cdot 0.58 + 0.5 \cdot 0.84 = 0.71 \end{aligned}$$

Voronoi approximation in $\mathbb{R}^2$

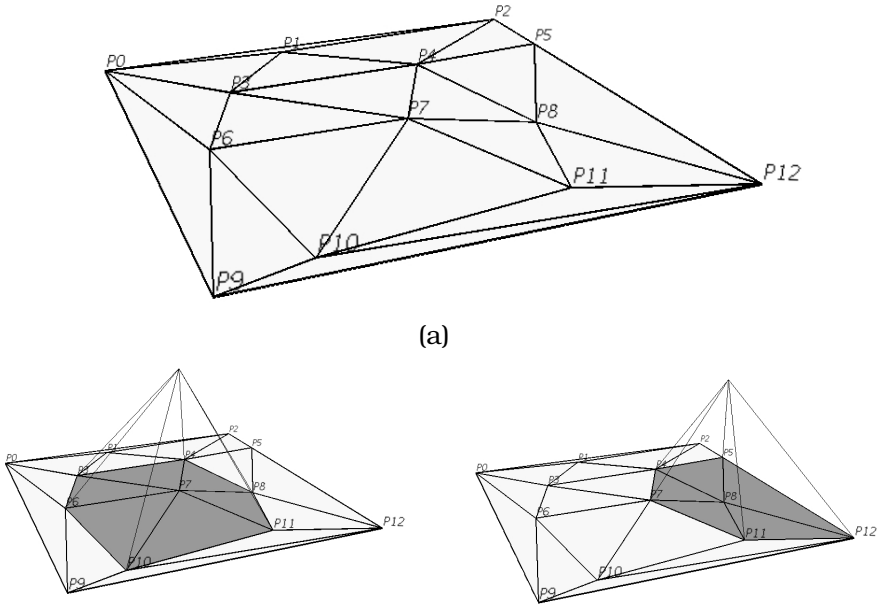
As an example, table 3.2 defines a set P of $N = 13$ points in $\mathbb{R}^2$, with their associated local approximators.

point	coordinates	local approximator
p_0	(-0.9,0.9)	$f_0(x_1, x_2) = 0 + x_1 + x_2$
p_1	(-0.2,0.85)	$f_1(x_1, x_2) = 0.1 - x_1 + x_2$
p_2	(0.8,0.92)	$f_2(x_1, x_2) = 0.2 + x_1 - x_2$
p_3	(-0.51,0.48)	$f_3(x_1, x_2) = 0.3 + x_1 + 2x_2$
p_4	(0.25,0.5)	$f_4(x_1, x_2) = -0.1 + 2x_1 - 3x_2$
p_5	(0.8,0.55)	$f_5(x_1, x_2) = 0.1 - 2x_1 + 2x_2$
p_6	(-0.7,0)	$f_6(x_1, x_2) = 0.1 + 0x_1 - x_2$
p_7	(0,0)	$f_7(x_1, x_2) = -0.2 + x_1 + 3x_2$
p_8	(0.4,-0.2)	$f_8(x_1, x_2) = -0.1 + x_1 - x_2$
p_9	(-0.9,-0.9)	$f_9(x_1, x_2) = 0 + 0x_1 + 0x_2$
p_{10}	(-0.6,-0.81)	$f_{10}(x_1, x_2) = 0 + x_1 + x_2$
p_{11}	(0.25,-0.7)	$f_{11}(x_1, x_2) = 0.1 + 2x_1 + 2x_2$
p_{12}	(0.8,-0.9)	$f_{12}(x_1, x_2) = 0.2 - x_1 + x_2$

Table 3.2: Set of points and local approximators used in the example.

Figure 3.10(a) shows the Delaunay triangulation defined by the set of points P. For simplicity reasons, the area defined by the external triangle is not shown. Figure 3.10(b) and 3.10(c) show the shape function defined for the Voronoi regions with center p_7 and p_8 respectively. The support of the shape function for p_7 (p_8) is shown in dark grey and corresponds to the union of all Delaunay triangles that have p_7 (p_8) as one of their vertices. Note that the value of the shape function is 1 at the center of the Voronoi region (i.e., just on the site) and goes down linearly to reach 0 in the neighbor sites (centers).

Given a point, the global approximation is computed as the linear combination of the evaluation of all local approximators that apply, weighted by the corresponding values of the shape function. It is important to stress that the local approximators are not evaluated directly with the coordinates, but with the local value of the coordinates defined by their difference with the corresponding center point. It means that give a point (x, y), the value of



(a)

(b)

(c)

Figure 3.10: Graphic plots of (a) the Delaunay triangulation defined by the points, (b) the shape function defined by p_7 and (c) the shape function defined by p_8.

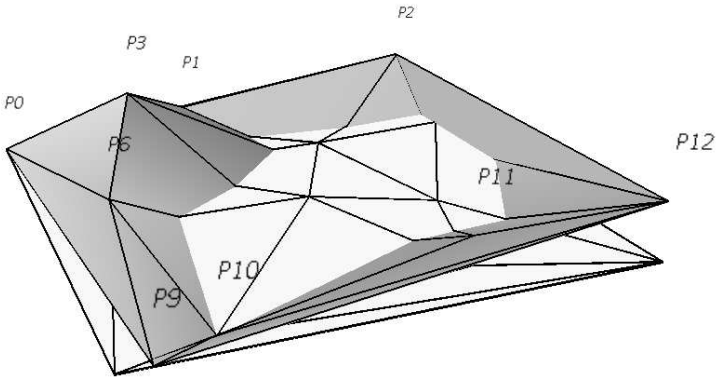
the local approximator f_i associated to the region with center $p_i = (x_{i1}, x_{i2})$ is computed as $f_i(x - x_{i1}, y - x_{i2})$.

As an example, let us compute the value of the Voronoi approximation for the point $a = (-0.9, 0.9)$. The summation of equation 3.5 reduces to a single term, since the point a is equal to p_0, the center of the Voronoi region $\mathcal{V}(p_0)$. The only shape function involved is ϕ_0, which evaluates to 1, since $\phi_0(p_0) = 1$ by definition. The local approximator f_0 is evaluated on the point $a - p_0$, which is $(0, 0)$ in this example, as shown below:

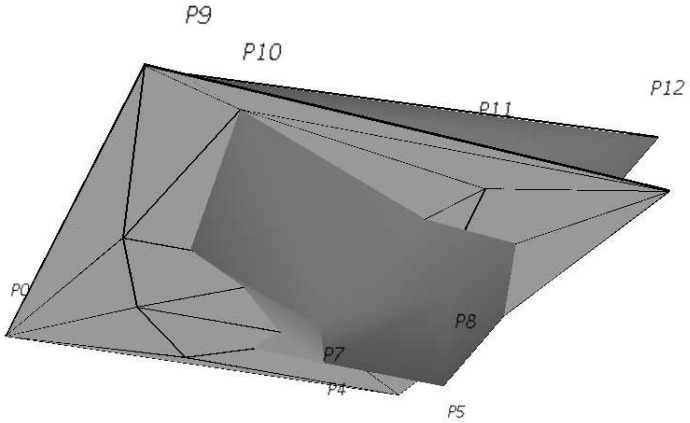
$$\begin{aligned} F(-0.9, 0.9) &= \phi_0(-0.9, 0.9) \cdot f_0(-0.9 + 0.9, 0.9 - 0.9) \\ &= 1 \cdot f_0(0, 0) = 1 \cdot 0 = 0 \end{aligned}$$

As other example, let us compute the value of the Voronoi approximation for the point $b = (0.2, -0.2)$. The point b is located in the Delaunay triangle defined by the vertices p_7, p_8 and p_{11}, meaning that the summation of equation 3.5 reduces to the three terms that involve the shape functions associated to these regions. The local approximators associated to the regions defined by the points p_7, p_8 and p_{11} are evaluated in the points $b - p_7$, $b - p_8$ and $b - p_{11}$ respectively. Note that the summation of $\phi_7(b) + \phi_8(b) + \phi_{11}(b)$ is 1, since they corresponds the the barycentric coordinates of b in the specified Delaunay triangle. The computation (restricted to 4 decimal places) can be summarized as follows:

$$
\begin{aligned}
F(0.2,-0.2) &= \phi_7(0.2,-0.2)\cdot f_7(0.2-0.0,-0.2-0.0)+ \\
&\quad \phi_8(0.2,-0.2)\cdot f_8(0.2-0.4,-0.2+0.2)+ \\
&\quad \phi_{11}(0.2,-0.2)\cdot f_{11}(0.2-0.25,-0.2+0.7) \\
&= 0.4327\cdot f_7(0.2,-0.2)+ \\
&\quad 0.3913\cdot f_8(-0.2,0.0)+ \\
&\quad 0.1739\cdot f_{11}(-0.05,0.5) \\
&= 0.4327\cdot -0.6+0.3913\cdot -0.3+0.1739\cdot 1.0=-0.2043
\end{aligned}
$$



(a)

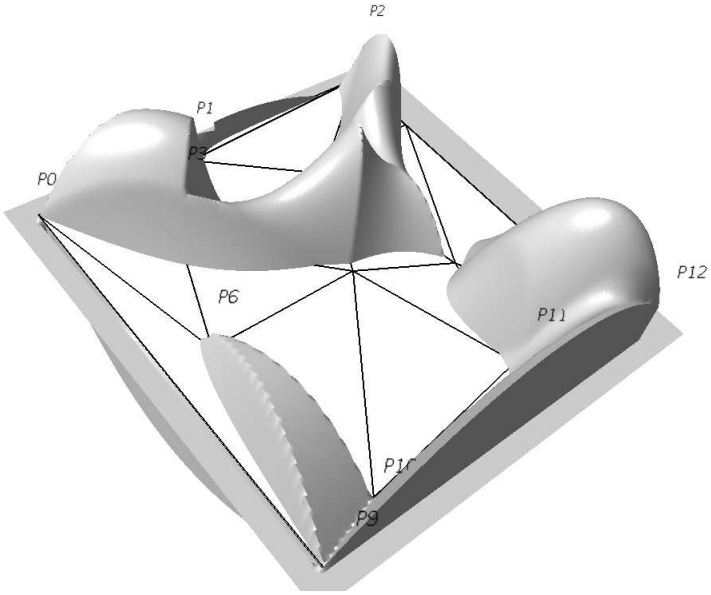


(b)

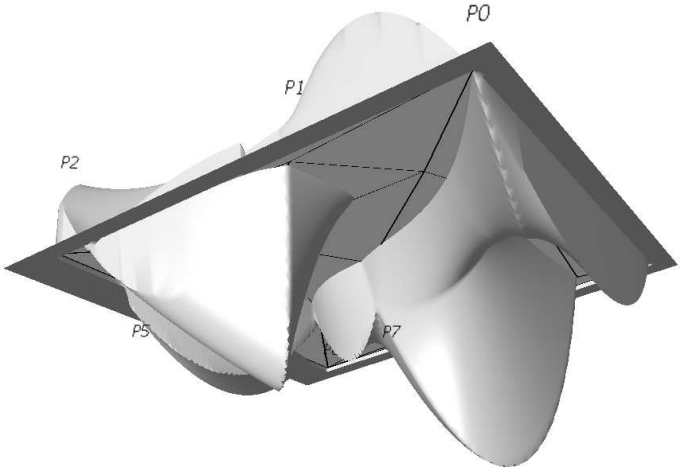
Figure 3.11: A (a) top and a (b) bottom view of the linear approximation defined by considering only the values of the Voronoi approximation in the points of P.

The plots in figure 3.11 show two views of the approximation defined as the linear combi-

nation of the local functions by considering only the values defined on the points of P. This is not the Voronoi approximation, it is just an example that shows the value that the function is expected to take in the centers of the Voronoi regions (or vertices of the Delaunay triangulation). The complete approximator is linear, since each local approximator behaves likes a constant function.



(a)



(b)

Figure 3.12: The complete Voronoi approximation in a view (a) from the top and (b) from the bottom. When comparing with the plots of figure 3.11 consider that a different view angle has been selected in order to improve visualization.

The plots in figure 3.12 show two views of the complete Voronoi approximation. Note that the value at the vertices are the same as in the plots of figure 3.11.

3.3 Evolution of Voronoi approximators

3.3.1 Representation

The representation is defined as a variable length list of Voronoi sites together with the associated linear local approximator parameters. Each Voronoi site is defined as a vector of real numbers of dimension d and the parameters of the local approximator as a vector of $d+1$ elements. Formally, an individual is a variable length vector:

$$\text{Ind} = \{R_1, \ldots, R_k, \ldots, R_N\}$$

where each component corresponds to a Voronoi site. Each component is defined as a vector:

$$R_k = \{p_k, a_k\} = \{(p_{k1}, \ldots, p_{kd}), (a_{k0}, \ldots, a_{kd})\}$$

where d is the dimension of the input space, a_{k0} and a_{ki} ($1 \leq i \leq d$) are the adjustable real valued parameters that define the local linear approximator and $p_k = (p_{k1}, \ldots, p_{kd})$ is the point in the input domain that corresponds to the center of the Voronoi region.

Figure 3.13 shows an example of a Voronoi approximation individual. Each individual defines a complete partition of the input space through Voronoi sites.

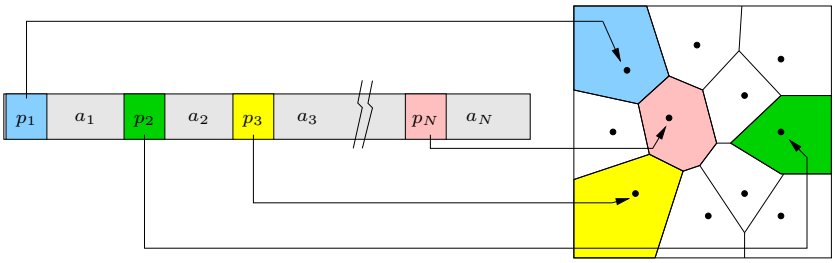


Figure 3.13: An example of the representation for an individual defined in $\mathbb{R}^2$.

An individual I is evaluated on a point x by using the algorithm detailed in pseudocode in listing 2.

The Delaunay triangulation defined by the points p_k is built by using a standard algorithm based on the computation of the convex hull, projected into the input space. Note that if the same individual is evaluated on different points, the Delaunay triangulation needs to be computed only once, since it does not depend on the particular point. The algorithm selects in line 3 the Delaunay region to which the point x belongs. The output value v is computed by following the standard procedure detailed in section 3.2. The shape function is computed for all Voronoi regions that have their center points p_k involved in the region r, since they are the only ones that will get non-zero values of the shape functions [1]. The value of the shape

[1] Note that the vertices of the Delaunay triangulations correspond to the centers of the Voronoi regions

1: **procedure** EVALUATION(I, x)
2: $d \leftarrow computeDelaunay(i)$ ▷ to be performed only once
3: $r \leftarrow selectDelaunayRegion(d, x)$
4: $v \leftarrow 0$
5: **for all** $vertex\ p_k\ in\ r$ **do** ▷ only active regions participate
6: $v = v + \phi_k(x) * f_k(x - p_k)$
7: **end for**
8: **return** v
9: **end procedure**

Algorithm 2: Pseudocode of the algorithm that computes the output produced by the Voronoi approximator represented by the individual I when evaluated at point x.

function defined by a Voronoi region at point x is computed by calculating the barycentric coordinate of x with respect to the points p_k (summits) of the Delaunay region to which the point belongs, as detailed in section 3.2.

3	0.89	0.05	7	0.42	−0.19	1	0.39	0.19	9	0.71	0.12

4	0.32	−0.06	8	0.4	0.07	7	0.32	0

Figure 3.14: An example of two individuals in $\mathbb{R}$.

As an example, figure 3.14 defines two individuals with four and three local approximators respectively in $\mathbb{R}$ ($d = 1$). The real values defined by the individuals are grouped in sets of three components. The first value of each set (of size d) is the coordinate of the Voronoi site while the second group of other values (of size d+1) corresponds to the parameters a_0 and a_1 of the local approximators $y = a_0 + a_1 x$. The first individual is defined with four regions with center points 1, 3, 7 and 9, and the four corresponding local approximators defined respectively with the parameters that follow each center point value. The second individual is defined with three regions with center points 4, 7 and 8.

Figure 3.15 shows a graphical representation of the two individuals defined in figure 3.14, displaying the partition of the input space, the shape functions, the local approximators and the complete approximation. Even if it is unusual to display Voronoi diagrams in $\mathbb{R}$, these examples have been selected since the concepts involved can be clearly identified with two dimensional graphics.

The diagrams labeled as *a* and *c* in figure 3.14 show the center points of each Voronoi region drawn over the *x*-axis, with the vertical lines defining the boundaries of the Voronoi regions, and the top arrows indicating the span of each one. The diagrams also display the shape functions as triangles that take the value 1 at the center of the Voronoi regions, 0.5 in the boundaries and 0 in the center of the neighbor Voronoi regions. The diagrams labeled as *b* and *e* show the local approximators defined by each Voronoi region. The diagrams labeled as *c* and *f* show the complete approximator defined by the individuals when the evaluation algorithm is applied.

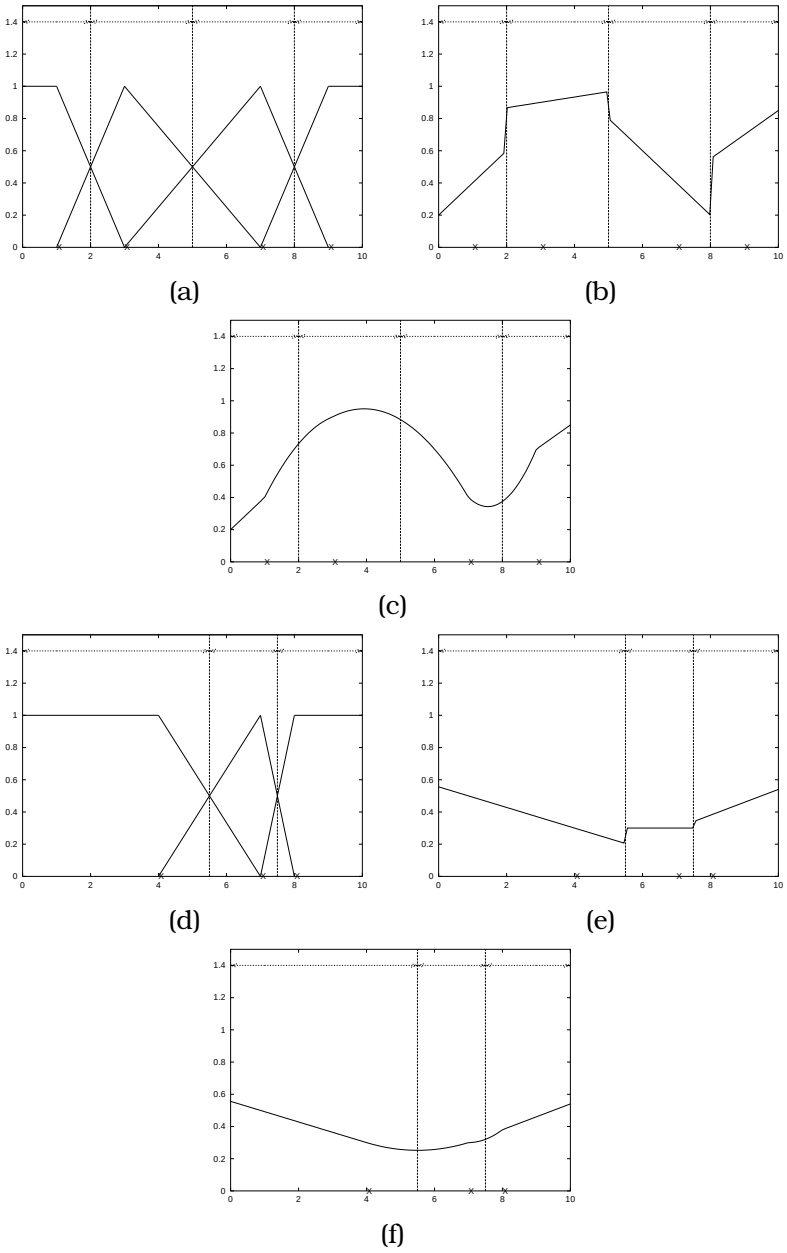


Figure 3.15: A graphical representation of the individuals defined in figure 3.14. The graph (a) shows the Voronoi regions and the shape functions, (b) the corresponding local linear approximators and (c) the complete approximation for the first individual. The graphs (d), (e) and (f) correspond to the second individual.

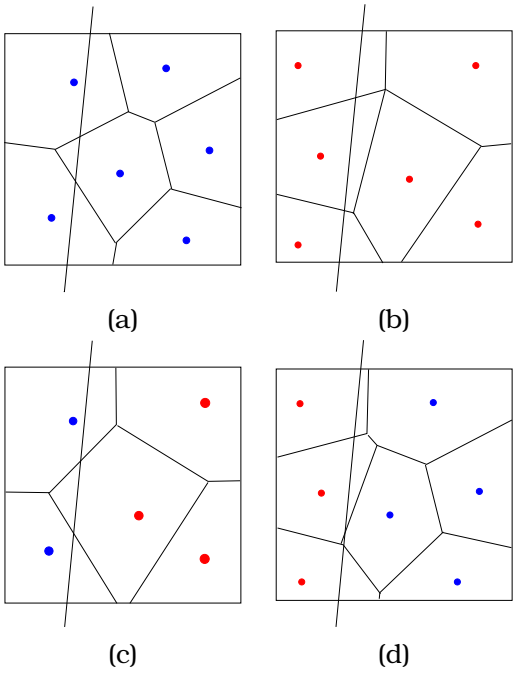


Figure 3.16: An example the effect of the Voronoi crossover in $\mathbb{R}^2$. The graphs (a) and (b) show the Voronoi diagrams defined by the parents and the graphs (c) and (d) the Voronoi diagrams defined by the children.

3.3.2 Recombination

The crossover operator performs a geometrically based crossover, by interchanging sites that belong to opposite areas divided by a random hyperplane (dimension $d-1$) [128] [82]. The first child is built by selecting the points (and the corresponding local approximator parameters) that are located at the left of the hyperplane in the first parent, and at the right of the same hyperplane in the second parent. The other child is defined with the remaining points, and the corresponding local approximator parameters. The concept of points being on the left or on the right of a hyperplane is defined with the usual semantics in computational geometry [31].

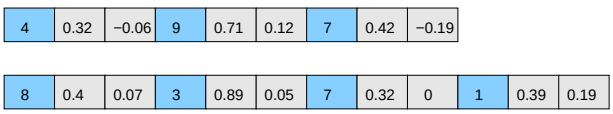


Figure 3.17: The two children obtained by performing a Voronoi crossover between the individuals defined in figure 3.14 with a hyperplane (a point in this case) defined as $x = 6$.

Figure 3.16 shows an example of the application of this crossover operator in $\mathbb{R}^2$. A line segment is defined randomly and used to separate the centers of the parents a and b. The first child c is defined by combining the points of the first parent that lie on the left of the segment with the points of the second parent that lie on the right of the same segment. The

other points are used to define the second child *d*. Note that the children can have a different number of Voronoi sites, when compared with the parents.

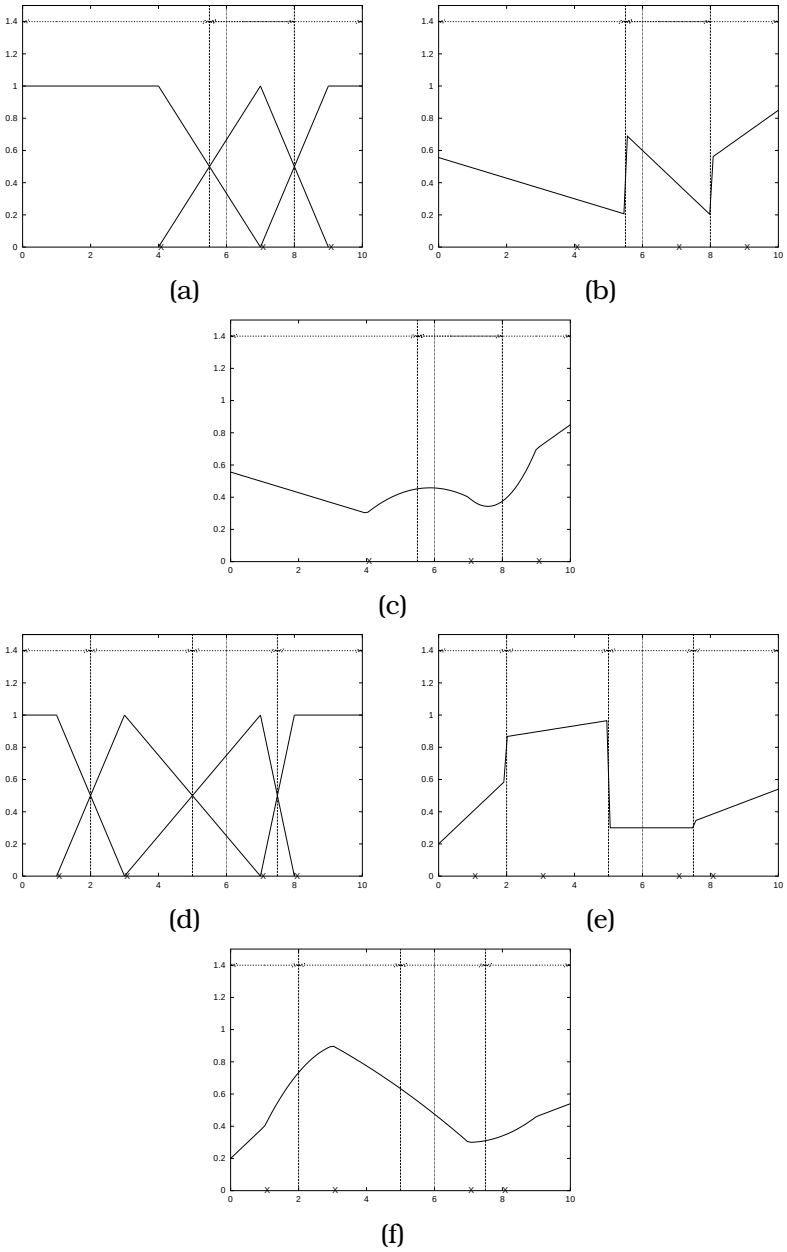


Figure 3.18: The first and the second child of the Voronoi crossover between the individuals defined in figure 3.14. The graphic (a) shows the Voronoi regions and the base functions, (b) the local approximators and (c) the complete approximation of the first child. The graphs (d), (e) and (f) correspond to the second child. The hyperplane is represented with the wider line at $x = 6$..

Figure 3.17 shows the two children obtained by performing a Voronoi crossover between the individuals defined in figure 3.14, with a hyperplane defined as $x = 6$. The random hyperplane corresponds to a single point in the input domain (since $d = 1$).

Figure 3.18 shows the graphical representation of the children. The graphs *a* and *d* show the points and the corresponding shape functions, the graphs *b* and *e* show the local approximators and the graphs *c* and *f* the complete approximation.

This operator was successfully applied to the problem of identification of mechanical inclusions [128] and other problems.

3.3.3 Mutation

Three types of mutations are applied during the evolution of Voronoi approximator individuals. The first type, which will be called *perturbation* is a standard mutation for real-valued individuals, which has a large probability to produce a small perturbation in the definition of the Voronoi diagram and/or the local approximators. The second and third mutation operators, that will be called *add* and *del* mutation respectively, are usually applied for variable length individuals and produce the addition and elimination respectively of a complete local approximator together with its associated Voronoi site. The three mutation types are specified below.

Perturbation

The purpose of this mutation operator is to generate one child from one parent by applying a small perturbation in one of the values of the parameters. It is implemented in terms of the zero-mean mutation or nonuniform mutation defined by Michalewicz et al. [107] (described in section 1.8, but repeated here for completeness). It is defined as follows:

$$x'_i(t) = \begin{cases} x_i(t) + \Delta(t, ub_i - x_i(t)) & if\, u < 0.5 \\ x_i(t) - \Delta(t, x_i(t) - lb_i) & if\, u \geq 0.5 \end{cases} \tag{3.10}$$

where $x_i(t)$ is the value of the parameter at time t, lb_i the lower limit of x_i, ub_i the upper limit of x_i and u is a random number uniformly generated in the range $[0, 1]$. $\Delta(t, y)$ is a function defined as follows:

$$\Delta(t, y) = y(1 - r^{(1-\frac{t}{T})^b}) \tag{3.11}$$

where r is a uniform random number in the range $[0, 1]$, T is the maximum generation number and b is the shape parameter (see equation 1.9). In all experiments performed in this thesis, the value of the parameter b is set to 5.

The effect of this mutation operator is to perform big exploratory steps in the beginning of the evolutionary process and small steps at the end. Two possible effects on a Voronoi approximator individual can be produced by this mutation operator, depending if the selected parameter is a Voronoi center coordinate or the parameter of a local approximator:

- *Voronoi coordinate*: When a site coordinate is selected for mutation, the complete Voronoi diagram can be slightly modified. The boundaries between the Voronoi region defined by

the selected site and its neighbor regions will be slightly updated. The local approximators defined in each region will not be modified, however, the complete approximation will change.

- *Local approximator parameter*: When a local approximator parameter is selected for mutation, the Voronoi diagram will remain unmodified, however, the complete approximation will be modified, but just in the region defined by the site that is associated to the local approximator.

Figure 3.19 shows an example of the application of this mutation operator to the first individual defined in figure 3.14, when a change is applied in a parameter that defines the second local approximator (the one that applies in the region with center $p_2 = 3$). The effect can be appreciated by comparing the diagrams labeled as (b) in figures 3.19 and 3.15 and the corresponding global approximation, in the area defined by the second local approximator.

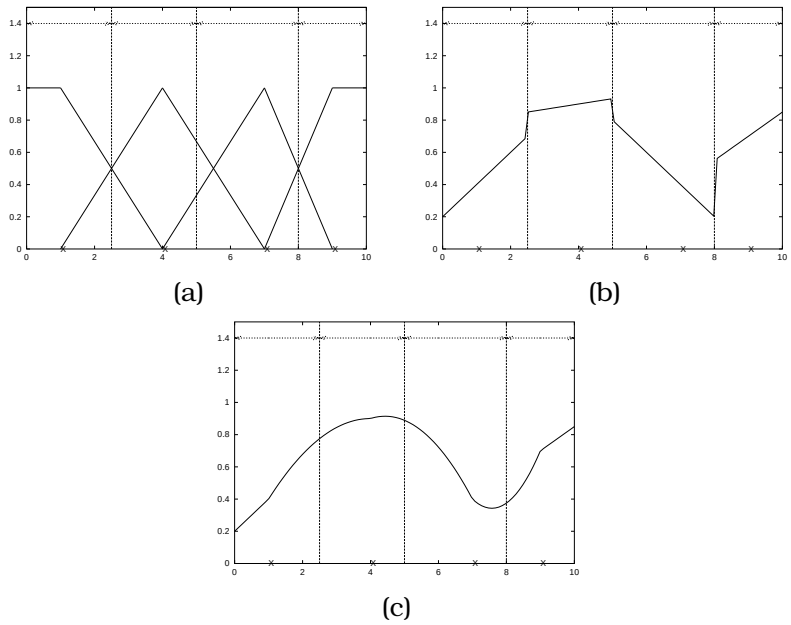


Figure 3.19: An example of the effect of the perturbation mutation operator when applied to the first individual defined in figure 3.14.

Add mutation

The second type of mutation adds a new Voronoi region together with a new local approximator to be applied in this region. Formally, an individual Ind $= \{R_1, \ldots, R_N\}$ with N local approximators is transformed into the individual Ind$' = \{R_1, \ldots, R_N, R_{N+1}\}$ with $N + 1$ local approximators by the addition of the new component:

$$R_{N+1} = \{p_{N+1}, a_{N+1}\} = \{(p_{(N+1)1}, \ldots, p_{(N+1)k}), (a_{(N+1)0}, \ldots, a_{(N+1)d})\}$$

where all parameters are defined randomly in the corresponding ranges.

As an example, figure 3.20 shows the effect of this mutation operator when applied to the first individual defined in figure 3.14, when a new Voronoi region and its corresponding local approximator are added with random parameters.

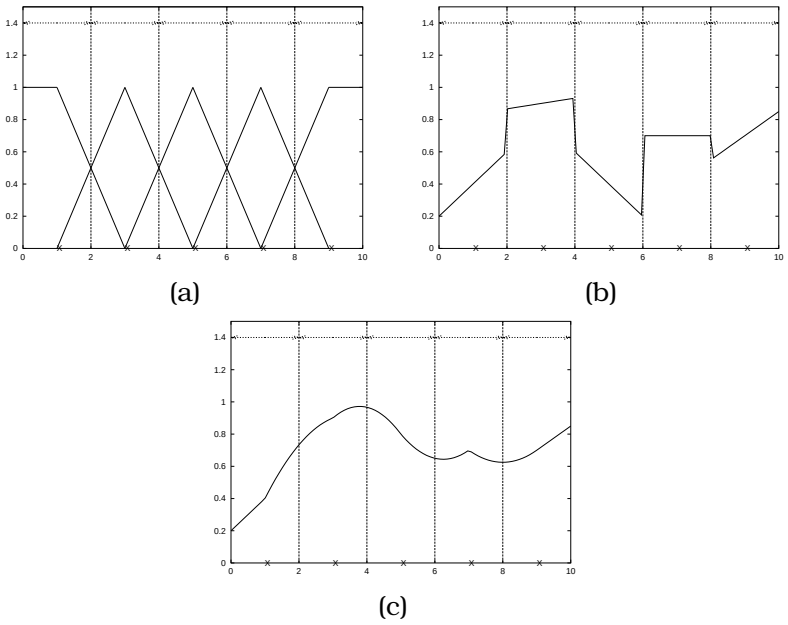


Figure 3.20: An example of the effect of the *add* mutation operator when applied to the first individual defined in figure 3.14.

Del mutation

The third type of mutation removes a Voronoi region together with its local approximator. Formally, an individual $\text{Ind} = \{R_1, \ldots, R_k, \ldots, R_N\}$ with N local approximators is transformed into the individual $\text{Ind}' = \{R_1, \ldots, R_{k-1}, R_{k+1}, \ldots, R_N\}$ with $N-1$ local approximators by removing the component $R_k = \{p_k, a_k\}$.

As an example, figure 3.21 shows the effect of this mutation operator when applied to the first individual defined in figure 3.14, when the third Voronoi region and its corresponding local approximator are removed.

3.3.4 Experimental Study of the Variation Operators

This section presents a study of the effect of different values of crossover and mutation probabilities on a function approximation problem solved by using the proposed representation and variation operators. The problem, and the quality of the solutions will be described in section 3.4.

The experiment is performed with a population size of 200. The initial population is created with random individuals with a length defined randomly between 3 and 100 local approximators each one. The experiment is repeated 120 times for 2000 generations, by performing 10

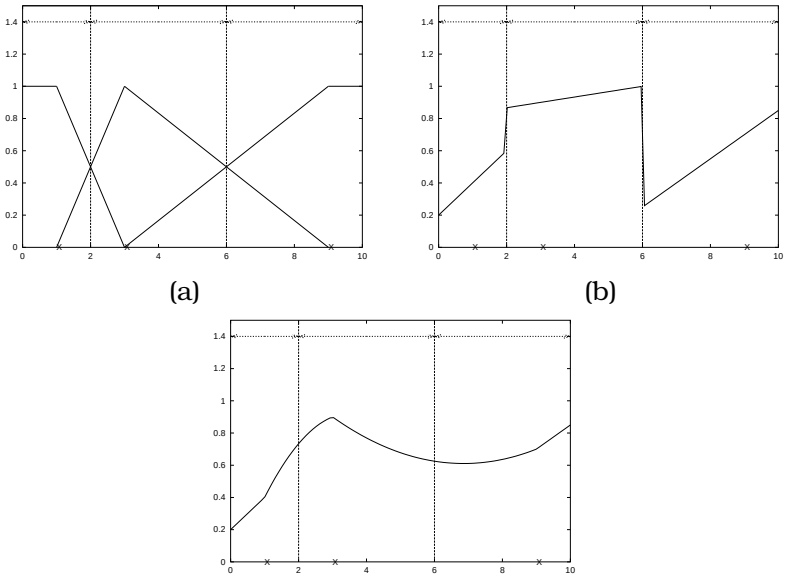


Figure 3.21: An example of the effect of the *del* mutation operator when applied to the first individual defined in figure 3.14.

runs of each one of the 12 combinations of the crossover probabilities 0.8, 0.6 and 0.4, and mutation probabilities of 0.8, 0.6, 0.4 and 0.2. Mutation probability is defined as the probability to apply one of the mutation operators. Once an individual has been chosen for mutation, the specific mutation operator that will be applied is chosen according to user-defined relative weights: the perturbation mutation is selected with a relative weight of 0.9, and the addition and deletion mutation operators with a relative weights of 0.05 each.

Table 3.3 summarize the results of the 120 runs, where each row corresponds to the average of the results of the 10 runs performed with the same set of parameters. In the table, *pc* stands for *probability of crossover*, *pm* for *probability of mutation* and *id* for identifier. For each set of parameters, the table presents the best, worst and average fitness obtained at the end of the evolutionary process. Also the size of the best individual and the average size of the population at the end of the evolutionary process are shown.

Figure 3.22 shows the plot with error-bars of the best individual fitness averaged for the 120 runs, plotted against the number of generations. The best, average and worst fitness obtained in the random initial population are 3,023,740, 3,319,810 and 3,410,060 respectively. The best, average and worst fitness obtained after the evolutionary process are 36,339, 52,062 and 73,821 respectively, demonstrating the robustness of the parameter setting. However, some settings give better results than others, as will be clear in the end of this chapter.

In order to keep this section to a reasonable size, the results of the 120 runs are summarized in four graphic plots.

Figures 3.23 and 3.24 present the fitness average values plotted against the number of generations in the 120 runs, organized by considering the crossover rates and the mutation rates respectively. Single lines in the graph of figure 3.23 correspond to average values of 40

id	pc	pm	fitness			size	
			average	best	worst	average	best
A	0.8	0.8	74096.85	36339.36	116363.00	49.09	29
B	0.8	0.6	74765.48	40836.10	100911.00	52.40	34
C	0.8	0.4	77777.48	56472.80	103895.00	52.79	25
D	0.8	0.2	80288.31	45474.50	104600.00	50.59	33
E	0.6	0.8	83460.84	51929.10	129248.00	44.20	21
F	0.6	0.6	70474.48	41507.10	96428.60	47.59	34
G	0.6	0.4	76455.99	47639.80	98118.60	45.70	39
H	0.6	0.2	92537.70	60654.19	129012.00	37.29	19
I	0.4	0.8	102235.04	65487.19	133847.00	21.79	9
J	0.4	0.6	83846.55	61336.60	110985.00	37.40	18
K	0.4	0.4	88358.00	43249.69	130950.00	34.79	13
L	0.4	0.2	95668.53	73821.89	117643.00	31.00	11

Table 3.3: Summary of the 120 runs. The results presented in each row correspond to the average results of the 10 runs.

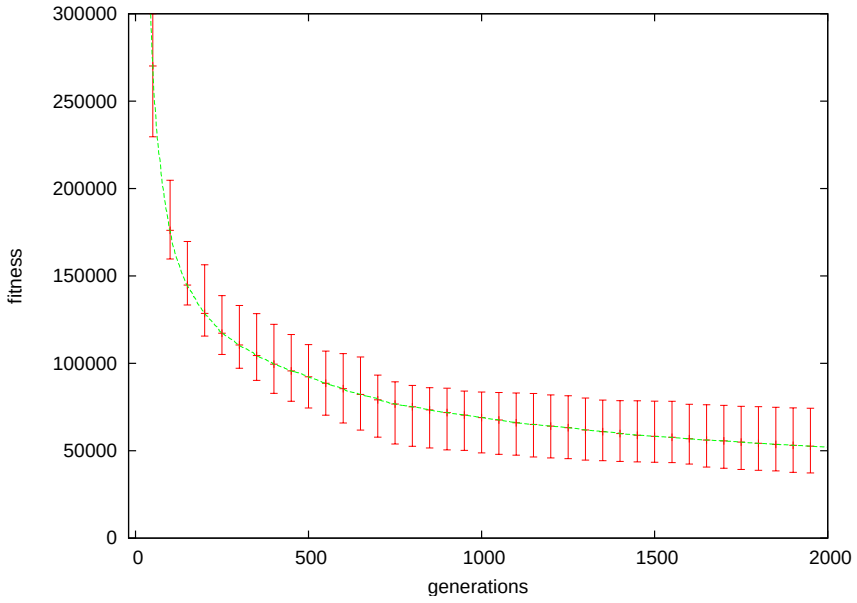


Figure 3.22: Best fitness in 120 runs of the experiment plotted with error-bars against the number of generations with different values of crossover and mutation probabilities.

runs performed with a single crossover rate and four different values for the mutation rate. The three lines correspond to the average of the runs labeled (see table 3.3) A to D, E to H and I to L respectively. Single lines in the graph of figure 3.24 correspond to average values of 30 runs performed with a single mutation rate and three different values for the crossover rate. The four lines correspond to the average of the runs labeled A-E-I, B-F-J, C-G-K and D-H-L

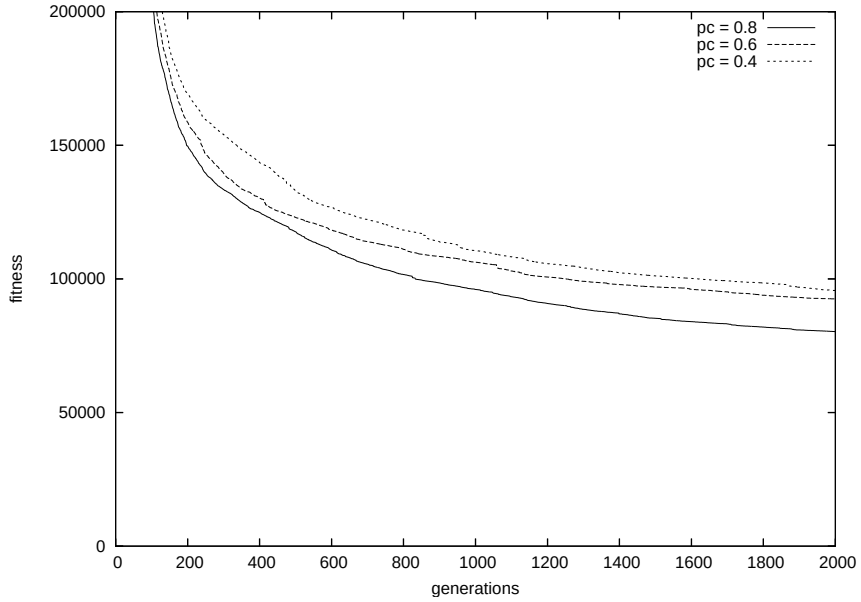


Figure 3.23: Average fitness plotted against the number of generations. Each line represents the average of 40 runs, 10 runs for each mutation rate (0.8, 0.6, 0.4 and 0.2).

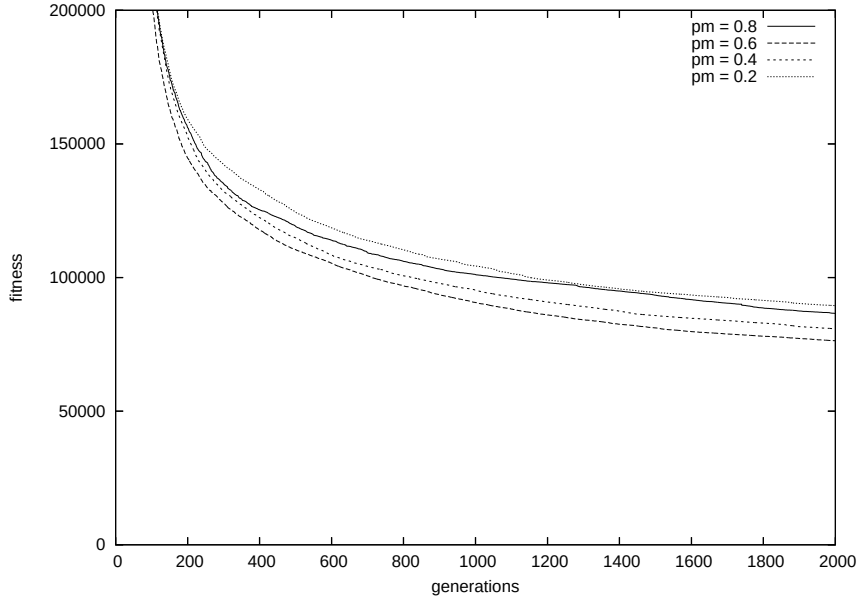


Figure 3.24: Average fitness plotted against the number of generations. Each line represents the average of 30 runs, 10 runs for each crossover rate (0.8, 0.6 and 0.4).

respectively. Please note that the results shown do not correspond to the best fitness values, but to average fitness values.

Figure 3.23 show that the worst fitness average values are obtained with the lowest crossover probability (0.4), and the best values with the highest crossover rate (0.8). Figure 3.24 show that best average results are found with a mutation rate of 0.6.

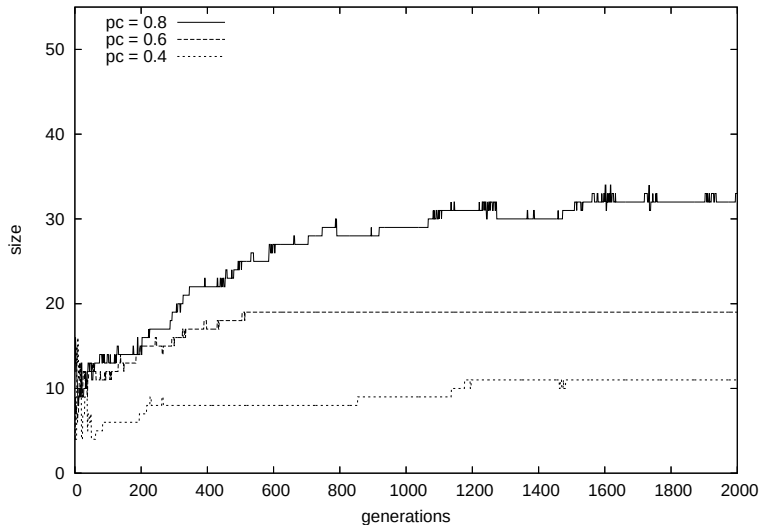


Figure 3.25: Average size of the best individual plotted against the number of generations. Each line represents the average of 40 runs with mutation rate of 0.8, 0.6, 0.4.

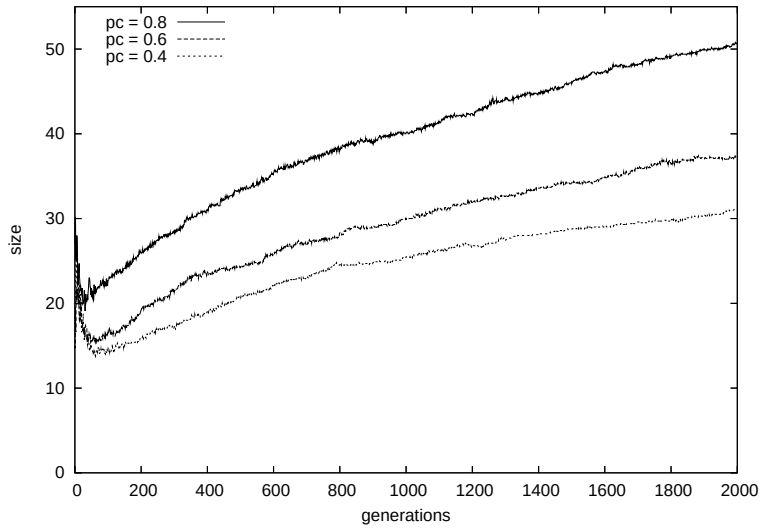


Figure 3.26: Average size of the individuals plotted against the number of generations. Each line represents the average of 40 runs with mutation rate of 0.8, 0.6, 0.4.

Figure 3.25 shows the size of the best individual plotted against the number of generations organized by crossover rate. Each single line corresponds to average values of 40 experiments,

organized by crossover rate. The plots show that higher crossover rates correspond to longer individuals, i.e., individuals with more local approximators. Figure 3.26 shows the average size of the individuals plotted against the number of generations organized by crossover rate. Each plot corresponds to average values of 40 runs. Again, higher crossover rates correspond to populations with longer individuals. The average size of the individuals of the population grows with the number of generations, however, the average size of the best individuals in the population reaches a stable value.

As a conclusion, higher values of crossover rates favor better solutions. In fact, the best individual was found with a crossover rate of 0.8, and the worst with a crossover rate of 0.4. Lower values of crossover rates can be used when solutions with a smaller number of local approximators are preferred, since the length of individuals is smaller by a factor of 2.

3.4 Function approximation

In this section, the evolution of Voronoi approximators is evaluated on two function approximation problems. The target functions are defined through a set of input output examples corrupted with noise. In the first problem, the set of examples is fixed and repeatedly presented for the evaluation of individuals in each generation. In the second problem, the examples are randomly selected in each generation, providing a learning scenario for incremental learning. The evolutionary algorithm has to address the problem of (1) selecting the number of local approximators, (2) determining their allocation in the input space and (3) selecting adequate values for the free parameters of each local approximator.

3.4.1 First experiment

The first experiment consists in the approximation of the function:

$$f(x,y) = e^{x/10}\sin^2(y/10) + e^{y/10}\sin^2(x/10) + N(0,1) \tag{3.12}$$

where N(0,1) is the noise generated with a Gaussian distribution centered at 0 with a deviation of 1. The function is characterized for a relatively large area with output value 0 and two narrow areas on the side with very high peaks that reach the value 6000, as shown in figure 3.27.

The input output examples used for the approximation algorithm are the 900 points corresponding to the vertexes of a 30x30 grid defined over the unit square.

The evolutionary algorithm for the evolution of the Voronoi individuals is run with the population size set to 200, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.6, with the perturbation relative mutation weight set to 0.9, and relative mutation weights for addition and removal of Voronoi sites set to 0.05 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 2000. During the evolution, the evaluation is performed by computing the error of the approximator in the 900 examples, with the output function perturbed with the Gaussian noise. Each evolutionary run of the experiment took about 20 minutes execution time.

Figure 3.28 shows the best approximation obtained with the Voronoi method in ten runs of the experiment, which visually confirms that the Voronoi evolution fulfills the expectations,

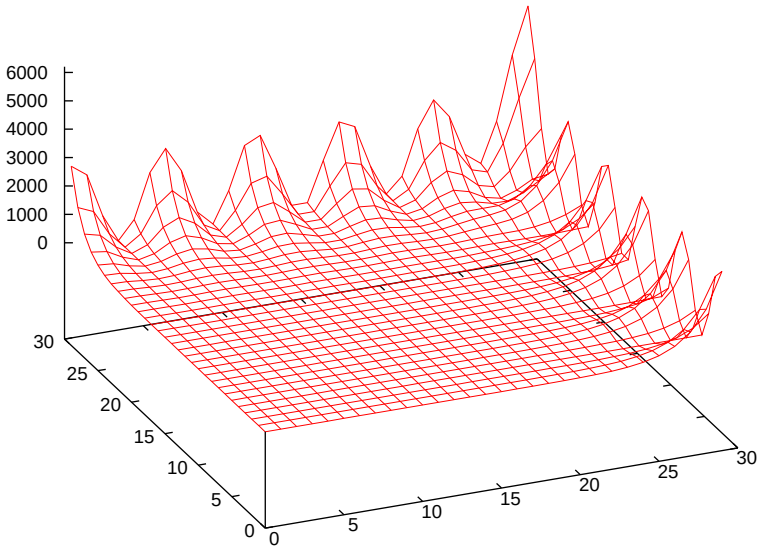


Figure 3.27: Target function for the first experiment.

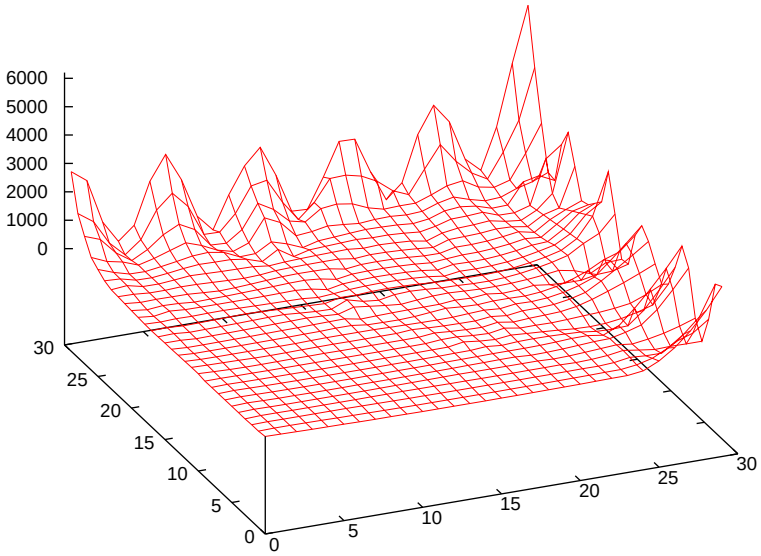
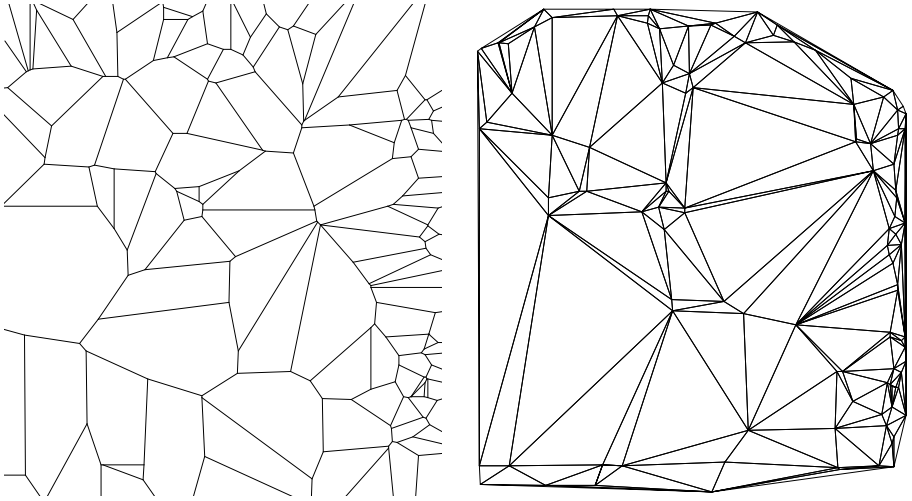


Figure 3.28: Best approximation obtained with the Voronoi approximation method.



(a) (b)

Figure 3.29: The (a) Voronoi partition and the (b) Delaunay triangulation defined by the best approximator obtained with the Voronoi approximation method.

providing a reasonable local based approximator for the function.

Figure 3.29 shows the Voronoi partition and the Delaunay triangulation defined by the best approximator obtained with the Voronoi approximation method. It can be appreciated that more local approximators are associated to areas that are more difficult to approximate, and few approximators to flat areas as expected.

As mentioned before, this function approximation problem was used to perform the experimental study of the variation operators described in section 3.3.4.

3.4.2 Second experiment

The second experiments consists in the approximation of a function which is defined as the composition of three simpler functions, with a low signal to noise relation and with a restricted number of examples presented for learning. The function to be approximated was proposed in [127] and it is defined as follows:

$$f(x,y) = max\{e^{-10x^2}, e^{-50y^2}, 1.25e^{-5(x^2+y^2)}\} + N(0,0.01) \tag{3.13}$$

where N(0,0.01) is the noise generated with a Gaussian distribution centered at 0 with a deviation of 0.01. Figure 3.30 shows a plot of the function, which consists of two perpendicular ridges with different wide, and a Gaussian bump at the origin.

The test set used to evaluate the quality of the approximation and the set of input output examples used by the approximation algorithm are defined in the same way as in [127] in order to compare the results. The test set consists in 1681 data corresponding to the vertexes of a 41x41 grid defined over the unit square, with the output values defined as the exact values of the function (without noise). The input output examples used for the approximation algorithm are defined as a sample of 500 points, drawn uniformly from the test set without replacement. The problem is enough complex to test the different algorithms.

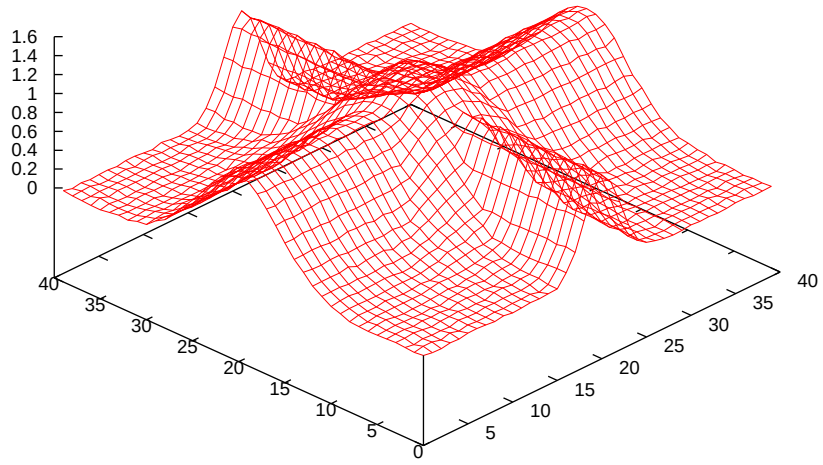


Figure 3.30: Target function for the second experiment.

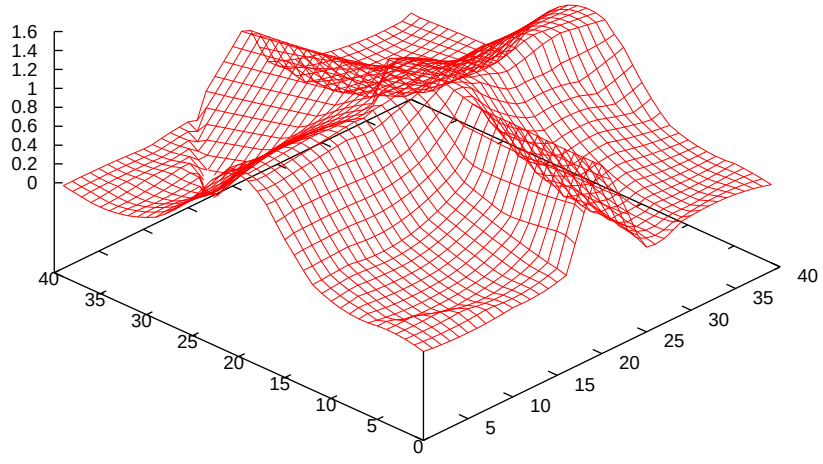
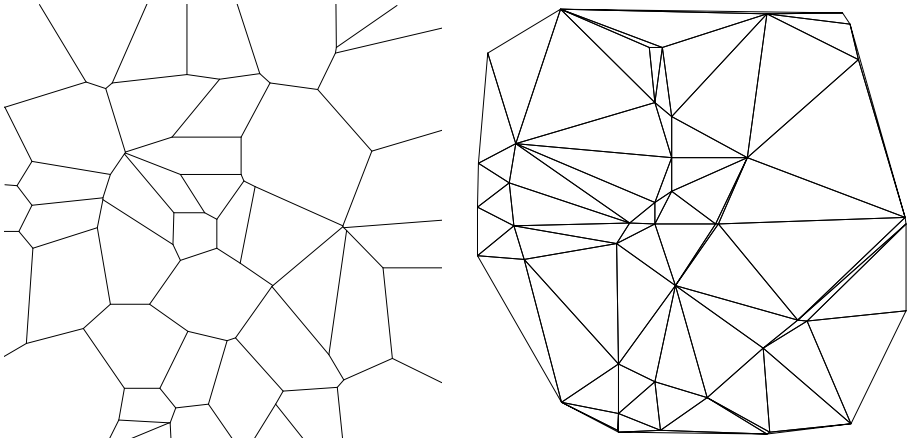


Figure 3.31: Best approximation obtained with the Voronoi approximation method.



(a) (b)

Figure 3.32: The (a) Voronoi partition and the (b) Delaunay triangulation defined by the best approximator obtained with the Voronoi approximation method.

The evolutionary algorithm is run with the population size set to 200, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.6, with the perturbation relative mutation weight set to 0.9, and relative mutation weights for addition and removal of Voronoi sites set to 0.05 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 2000. The fitness is defined as the normalized mean square error (NMSE) [61]. During the evolution, the evaluation is performed by computing the error of the approximator in 500 randomly selected examples, with the output function perturbed with the Gaussian noise. Each evolutionary run of the experiment took about 65 seconds execution time.

Table 3.4: NMSE error for the function approximation experiment for linear regression (LR), sigmoidal three layer backpropagation neural network (NN), hierarchical mixture of experts (HME), receptive field weighted regression (RFWR) and Voronoi approximation (VA).

LR	NN	HME	RFWR	VA
1	0.04	0.04	0.02	0.029

The results are compared with the results of four other algorithms, as presented in [127]. The algorithms are a standard global MSE-based linear regression (LR) and three gradient-based learning algorithms: sigmoidal three layer back-propagation neural network (NN), mixture of experts model (HME) and receptive field weighted regression (RFWR). The neural network was trained with the back propagation algorithm using 20 to 100 hidden units and sigmoid units, for 20000 epochs. The mixture of experts was trained with the expectation-maximization (EM) algorithm, with 25, 50, 75 and 100 experts and the weighted regression algorithm was initialized with 16 evenly distributed local approximators. Table 3.4 show the best results obtained by the different methods when evaluated on the test function. The performance of the Voronoi approximation method is acceptable, considering that the NN, HME and RFWR are gradient-based learning algorithms, which can use the derivative information

in order to update the free parameters.

Figure 3.31 shows the plot of the best approximation obtained with the Voronoi approximation method, in ten runs of the experiment. Figure 3.32 shows the Voronoi partition and the Delaunay triangulation defined by the best approximator obtained with the Voronoi approximation method. The distribution of the sites is almost regular when compared with the previous experiment, where there were very clearly identified areas of different complexity in the shape of the target function.

3.5 Conclusions

This chapter has introduced the Voronoi approximation method, a mesh-less approximation defined in terms of geometric concepts. The Voronoi approximation is built by partitioning the input space into a set of regions, and defining a set of functions to be applied in each region. In this way, instead of the solution being defined as a single *global* function, it is defined by the combination of a set of *local* functions. In this sense, it is comparable to many other local approximation strategies. However, the main contribution of the approach, is the interesting representation for evolutionary algorithms, where each local approximator can be represented by a single point (which define the region), and a set of parameters (which define the local function). The representation, together with the properties derived from the definition of the shape functions using Voronoi and Delaunay concepts, allow the use of genetic operators defined in geometric terms.

The results presented in this chapter are a proof-of-concept of the feasibility using evolutionary algorithms to tune Voronoi sites to perform function approximation. Nevertheless, those results were obtained on data-fitting problems, and demonstrate that the efficiency of the proposed evolutionary approach is competitive with that of other approaches. However, it should be kept in mind that the computing time needed by this evolutionary approach remains larger than that of some other approaches, meaning that in data-fitting context, the proposed approach is not competitive. Next chapter will introduce another context in which the situation is very different: whereas the problem can be seen as function approximation too, no examples are available, making most data-fitting-oriented methods unapplicable.

Chapter 4

Voronoi-based Fuzzy Systems

This chapter introduces a method to define fuzzy systems based on the Voronoi approximation presented in the previous chapter. This method allows the definition of Takagi Sugeno fuzzy systems by partitioning the input space in Voronoi regions, and assigning multivariate membership functions to each one of these regions. The fuzzy rules use these membership functions in the antecedents and have local linear approximators as consequents.

The chapter is structured as follows. Section 4.1 introduces the Voronoi-based fuzzy system and section 4.2 discusses their main properties. Section 4.3 describes the evolutionary approach used to evolve Voronoi-based fuzzy systems and section 4.4 presents experimental results on two control problems.

4.1 The Voronoi-based Fuzzy System

This section presents a method to define fuzzy systems in terms of the Voronoi approximation [83]. It will be shown that it is not necessary to modify the definition of the Voronoi approximation, but just to provide a different semantic to the elements that define it. The so called Voronoi-based fuzzy system, that is going to be defined in this section, belongs to the class of Takagi Sugeno fuzzy systems, and its main particularity is that the fuzzy membership functions are multidimensional, something that allows to define a non regular partition of the input space.

Definition 22 *Given a set of ω points $P = \{p_1, \ldots, p_\omega\}$, a MISO Voronoi-based fuzzy system is a Takagi-Sugeno fuzzy system where the input space is partitioned in the ω regions defined by the Voronoi diagram induced by P and the fuzzy membership functions used in the antecedents of the ω rules correspond to the shape functions defined by the Voronoi approximation.*

A MISO [1] Voronoi-based fuzzy system in $\mathbb{R}^n$ has n input variables organized as an input vector $x =< x_1, x_2, \ldots, x_n >$ and one output variable v. The input vector x is fuzzified by ω fuzzy membership sets A_i ($1 \leq i \leq \omega$) whose multivariate membership functions μ_i ($i = 1, 2, \ldots, \omega$)

[1] Note that as discussed in section 2.4.3, a MIMO fuzzy system can be considered as a set of MISO fuzzy systems. Hence, all definitions in this section just consider MISO Voronoi-based fuzzy systems without loss of generality.

are defined in terms of the Voronoi approximation. The output variable is defined as a linear combination of the input variables.

As an example, the definition of the k-th ($1 \leq k \leq \omega$) rule R_k follows:

$$if\ x\ is\ A_k then\ v_k = a_{0k} + a_{1k}x_1 + \ldots + a_{nk}x_n \tag{4.1}$$

where A^k is the multivariate antecedent fuzzy set associated to the input vector x, v_k is the value of the output variable v defined by the k-th rule, and a_{0k} and a_{ik} are adjustable real valued parameters ($1 \leq i \leq n, 1 \leq k \leq \omega$).

Note that the definition of a rule in a Voronoi-based fuzzy system is similar to the definition of an equivalent rule in a standard Takagi Sugeno fuzzy system (see section 2.4.2). The only difference is that there is a single multidimensional fuzzy membership function used in the antecedent instead of a set of fuzzy membership functions associated to each one of the input variables.

Following the standard approach in fuzzy system model presentations, figure 4.1 shows a schematic diagram of the Voronoi-based fuzzy system, structured in four layers. The first layer corresponds to the input layer, the second to the antecedent fuzzy memberships, the third to rules and the fourth to outputs.

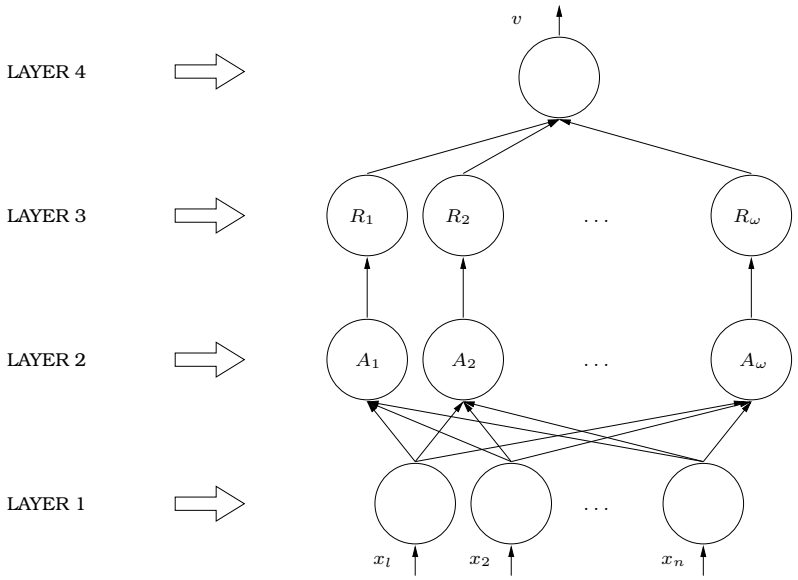


Figure 4.1: The structure of the Voronoi-based fuzzy system

Units in the first layer are called input units, each one corresponding to one of the inputs of the fuzzy controller. Units in the second layer are called partition units. They act as multidimensional fuzzy membership functions. Units in the third layer are called rule units. Each fuzzy rule in the Voronoi-based fuzzy system has a corresponding rule unit. Note that there is a one to one correspondence with units in the second layer, since there is a single multidimensional fuzzy set associated to each rule. Units in the fourth layer are called output units. They compute the outputs as a weighted linear combination of input units, generating the external output.

The function of each type of unit is described below.

Layer 1 : No computation is performed in this layer. The external input vector x is transmitted to the units in layer 2.

Layer 2 : The k-th unit in this layer computes the fuzzy membership value $\mu_{A_k}(x)$ of the input vector x to the multidimensional fuzzy set A_k associated to the k-th rule.

Layer 3 : Units in this layer compute a linear combination of input values based on the parameters specified by each rule, weighted by the corresponding degree of activation, as usual in Takagi Sugeno fuzzy systems. The output v^k produced by the unit k is:

$$v^k = (a_{k0} + \sum_j a_{kj}(x_j - p_j^k))\mu_k(x) \tag{4.2}$$

where the a_{kj} are the real valued parameters that compute the linear combination of input values associated to the rule k and p_{kj} is the j-th component of the site p_k, center of the Voronoi region associated to A_k.

Layer 4 : Units in this layer compute the output v by computing the summation of the corresponding outputs produced by each rule. That is:

$$v = \sum_k v^k \tag{4.3}$$

Note that a Voronoi-based fuzzy system is just a Voronoi approximation, with a different semantic associated to the elements involved in the definition (see table 4.1). The points (or sites) of the Voronoi approximation correspond to the centers of the multivariate membership functions of the antecedents, the shape functions correspond to the multivariate fuzzy membership functions and the local approximators to the consequent linear approximators. The evaluation of the Voronoi-based fuzzy system is performed in the same way as the Voronoi approximation.

Voronoi approximation	Voronoi-based fuzzy system
sites	centers of antecedent membership functions
shape functions	antecedent membership functions
local approximators	consequent local approximators

Table 4.1: Equivalence of the elements between a Voronoi approximation and a Voronoi-based fuzzy system.

The multivariate fuzzy set defined in terms of the shape functions of the Voronoi approximation, coincides with the definition of the membership functions of the meaning of the symbols defined in the context of fuzzy sensors [19] by Benoit et al. in 1994. In their approach, the input space (called measurement space) is partitioned by a Delaunay triangulation, and the membership functions associated to the symbols are defined on each simplex with multi linear interpolation [18], producing by a different approach, the same hat-like function.

Next sections will discuss a number of interesting properties of Voronoi-based fuzzy systems that highlights the differences with previous approaches.

4.2 Properties

The Voronoi-based fuzzy system belongs to the class of approximative fuzzy models [8], where each fuzzy rule defines its own fuzzy sets. It provides continuous output, as most fuzzy systems, and it also has a number of useful properties, that we shall now discuss in turn.

4.2.1 Fuzzy finite partition

The multivariate fuzzy membership functions defined by a Voronoi-based fuzzy system define a finite fuzzy partition in the input space. As it was explained in section 2.2, finite fuzzy partitions of the input space are desirable in order to allow a uniform and complete coverage of the input space. Figure 4.2 shows the multivariate membership functions defined by the set of 13 points used in the example of section 3.2.1. Note that, without considering open Voronoi regions, the complete input space is covered with membership functions. External membership functions, i.e., membership functions that involve external points (which are not shown in the figure for visualization purposes) cover the outer space.

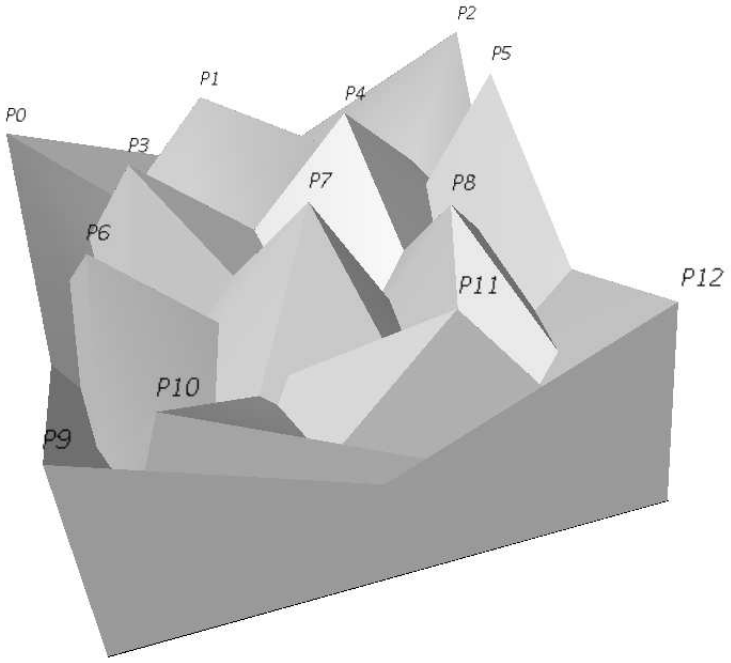


Figure 4.2: The membership functions defined by the same set of point used in the example of section 3.2.1.

Theorem 1 *The multivariate fuzzy membership functions defined by a Voronoi-based fuzzy system in* $\mathbb{R}^2$ *define a finite fuzzy partition in the input space.*

Demonstration:

In order to show that the membership functions define a finite fuzzy partition (see section 2.2) it is necessary to show that (1) the membership functions are normal and (2) the summation of all membership functions for a single point in the input domain is always 1.

Step 1:

Membership functions in a Voronoi-based fuzzy system are normal by definition, since their value at the center point is 1, as shown in section 3.2.

Step 2:

It is necessary to prove that:

$$\forall x \in U,\ \sum_i \mu_{A_i}(x) = 1$$

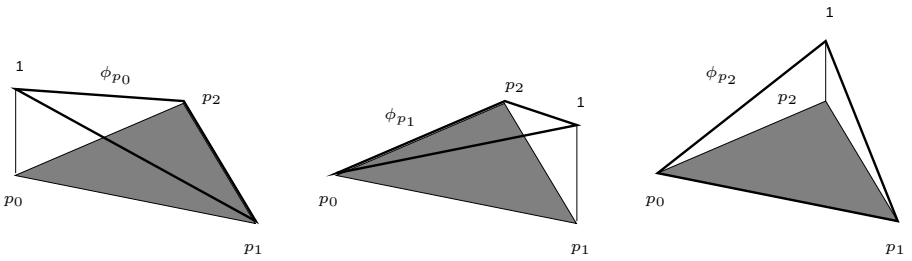
Given a set of points $P = \{p_1, \ldots, p_N\}$ in $\mathbb{R}^2$ and the set of external points Q, by definition of the Delaunay triangulation, every point $x \in U$ belongs to either:

- a single Delaunay region $\mathcal{T}$ defined by three points p_1, p_2 and p_3.

or:

- two or three Delaunay regions respectively, if the point is located over an axe (triangle boundary) or is a vertex.

In the last case, one of the involved triangles is randomly selected. By following this assumption and considering the definition of the shape functions, the point x belongs to exactly three membership functions: the shape functions defined by the points p_0, p_1 and p_2, which are all of them centers of Voronoi regions. Figure 4.3 shows an example of the involved side of the membership functions that are defined over the selected triangle.



$\mu(p_0) = 1$ $\mu(p_1) = 1$ $\mu(p_2) = 1$

Figure 4.3: The three shape functions defined over a Delaunay triangle in $\mathbb{R}^2$.

The value of the membership function for x is defined as the relative area of the sub-triangle defined by the point x and the points of the triangle excluding the center of the region. Lets call A_0, A_1 and A_2 respectively the area of the triangles defined by x and p_1 and p_2, x and p_0 and p_1 and x and p_0 and p_2. By considering A as the area of the triangle $\mathcal{T}$, the membership values are:

$$\mu_{p_0}(x) = \frac{A_0}{A}\ \ \mu_{p_1}(x) = \frac{A_1}{A}\ \ \mu_{p_2}(x) = \frac{A_2}{A}$$

Note that these areas are complementary, and their summation is 1 by definition of barycentric coordinates:

$$\mu_{p_0}(x) + \mu_{p_1}(x) + \mu_{p_2}(x) = \frac{A_0 + A_1 + A_2}{A} = \frac{A}{A} = 1$$

The same reasoning can be applied for higher input dimensions.

4.2.2 ϵ-completeness property

The ϵ-completeness property, defined in section 2.2, establishes that any input must belong to at least one fuzzy set with a membership value not smaller than a threshold ϵ. This property guarantees an adequate representation for every input point, since there is always a rule that is applied with at least a known value of membership.

Theorem 2 *Voronoi-based fuzzy systems in* $\mathbb{R}^2$ *fulfills the* ϵ*-completeness property for* $\epsilon = \frac{1}{3}$*.*

Demonstration:

Immediate by considering that the membership functions are defined in terms of barycentric coordinates. The minimum value of the barycentric coordinates of the point x, by considering the three coordinates at the same time, is when the point x is exactly at the center of the triangle, with barycentric coordinates with value $\frac{1}{3}$ (one third of the relative area for each sub-triangle).

The same reasoning can be applied for higher input dimensions $d > 2$, where the corresponding ϵ value is $\frac{1}{(d+1)}$. Note that by changing the definition of the shape function ϕ is possible to reach other required ϵ values.

4.3 Evolution of Voronoi-based fuzzy systems

Since a Voronoi-based fuzzy system is a Voronoi approximator, the same evolutionary approach, including representation and operators, can be used to evolve Voronoi-based fuzzy systems. The only difference is that support for MIMO fuzzy systems has to be included, something that can be provided with a very simple extension of the original representation. Section 4.3.1 presents details on the representation proposed.

However, the most important point to be considered is that a set of new and very appealing properties for fuzzy systems arise due to the extra semantic that was added in the proposed model. These properties are discussed in section 4.3.2.

4.3.1 Representation

As mentioned before, since a Voronoi-based fuzzy system is a standard Voronoi approximator, the same representation proposed in section 3.3.1 can be used for evolution. However, in the context of fuzzy systems, approximators with multiples outputs (or MIMO fuzzy systems) should also be considered. Since a MIMO fuzzy system can be represented as a set of MISO fuzzy systems (see section 2.4.3), the extension of the representation to include MIMO support is immediate.

Formally, an individual is defined as a variable length vector:

$$\text{Ind} = \{R_1, \ldots, R_k, \ldots, R_N\}$$

where each component corresponds to a Voronoi site. Each component is defined as a vector:

$$R_k = \{p_k, a_k^1, \ldots, a_k^m\} = \{(p_{k1}, \ldots, p_{kd}), (a_{k0}^1, \ldots, a_{kd}^1), \ldots, (a_{k0}^m, \ldots, a_{kd}^m)\}$$

where d is the dimension of the input space, m is the number of output variables, $a_k^j = (a_{k0}^j, \ldots, a_{kd}^j)$ are the adjustable real valued parameters that define the local linear approximator associated to the output variable j ($1 \leq j \leq m$) and $p_k = (p_{k1}, \ldots, p_{kd})$ is the point in the input domain that corresponds to the center of the Voronoi region.

Figure 4.4 shows an example of a Voronoi based fuzzy system individual.

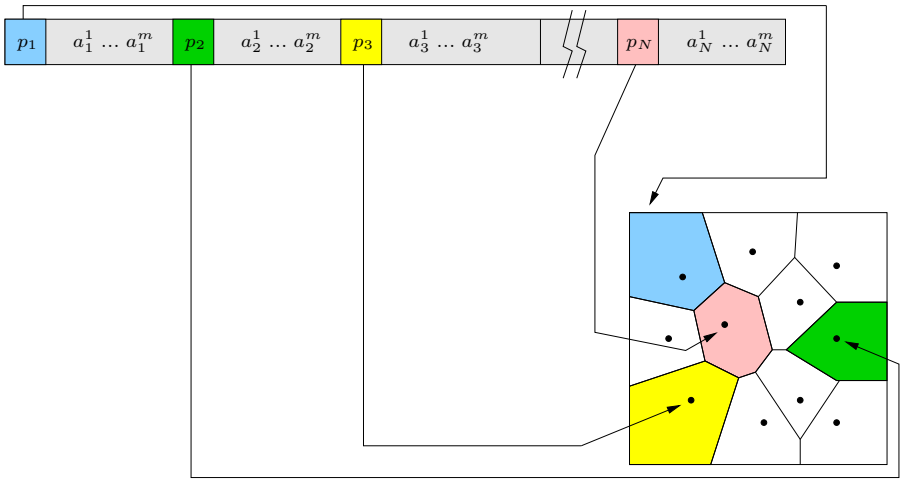


Figure 4.4: An example of the representation for a Voronoi-based fuzzy system individual defined in $\mathbb{R}^2$ with m outputs.

The evaluation of the individuals is performed with the same algorithm used for the evaluation of Voronoi approximators (see algorithm 2 in section 3.3.1), performing multiple evaluations for the different outputs if necessary.

4.3.2 Properties

The set of rules in a Voronoi-based fuzzy system is not a set of independent rules. There is a synergic relation between the rules which is particularly valuable in the context of evolutionary algorithms. The variation operators can be thought as operating over these relations, changing the interaction between the rules.

Other interesting property relates to the possibility to incorporate *a priori* knowledge before the evolution. This is not a new concept, and in fact it was considered in some of the evolutionary approaches described in section 2.6. However, the Voronoi-based fuzzy systems allow the inclusion of *a priori* knowledge in a very simple way, including knowledge about the

solution in specific points in the input domain. Moreover, this knowledge is *naturally* refined by the algorithm.

Both properties are discussed in detail below.

Adaptive fuzzy rules

As a very unusual property for fuzzy systems, the influence on the output of a particular fuzzy rule in a Voronoi-based fuzzy system does not only depend on the rule itself, but also depends on all neighbor rules. As a consequence of that, the area of application of a single rule is automatically modified when a new neighbor rule is inserted, removed or even modified.

Formally, the area of application $\mathcal{A}(R_k)$ of a fuzzy rule R_k is defined as the union of all Delaunay regions which contain the point p_k, center of the rule R_k:

$$\mathcal{A}(R_k) = \bigcup_{p_k \in D_j} D_j \quad D_j \in D = \{D_1, \ldots, D_\gamma\}. \tag{4.4}$$

where p_k is the center of the rule R_k and $D = \{D_1, \ldots, D_\gamma\}$ is the Delaunay partition of the set $\mathcal{P} = \{p_1, \ldots, p_N\}$.

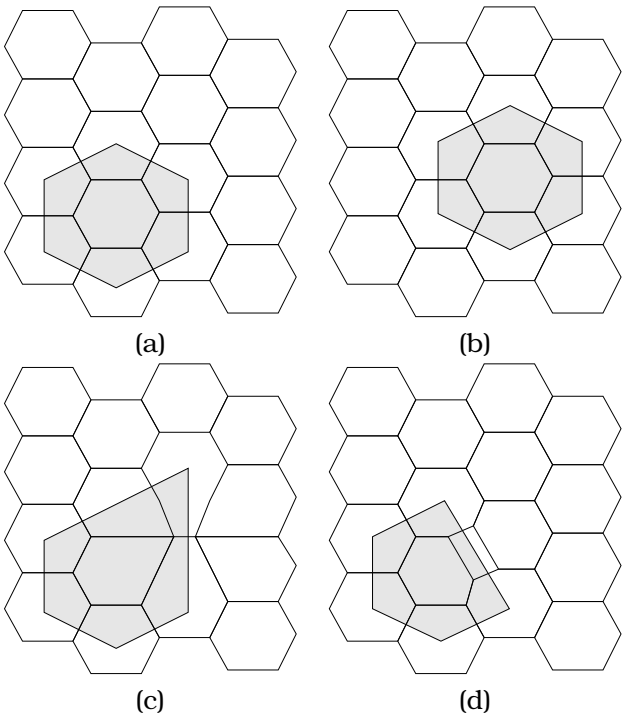


Figure 4.5: Diagram (a) shows the application area of a fuzzy rule. Diagram (b) shows the application area of one of its neighbor rules. Diagram (c) shows the application area of the rule of diagram (a) when the rule of diagram (b) is removed, and diagram (d) the application area of the rule of diagram (a) when a rule is added between both rules.

Figure 4.5 shows an example of the application area of some rules in a regular partition,

and illustrates the interdependency of application areas of neighbor rules when some of them are removed or added.

The evolutionary algorithm evolves individuals that represent complete fuzzy systems defined by a set of fuzzy rules that are synergistically related, and not fuzzy systems defined with a set of independent fuzzy rules. The variation operators hence modify the application areas of all fuzzy rules, while still maintaining the required ϵ-completeness level.

Adaptive *a priori* rules

One of the advantages of fuzzy systems is the possibility to use *a priori* knowledge, a well known concept that specifies the information about the desired form of a solution which is additional to the information provided by the training data [22]. The correct use of *a priori* knowledge during model design leads to better models even in the presence of deficient and incomplete data sets [143].

In some evolutionary based fuzzy systems design approaches, the user can incorporate *a priori* knowledge by manually defining fuzzy sets and the corresponding fuzzy rules. This initialization implies that some restriction on the output values and the partition of the input space is introduced in the evolutionary process, but the expected benefit is that the evolutionary algorithm, biased toward hopefully good parts of its search space, will converge faster to better solutions.

Similarly, the Voronoi-based fuzzy system allows the definition of *a priori* rules, i.e. fixed Voronoi sites with associated local approximators that will not be modified by evolution. But one big advantage of the Voronoi-based fuzzy system is that the expert does not need to specify the application area of such rules: thanks to the synergistic effect described before, the evolutionary process, by adding rules more or less close to the *a priori* rules, will also tune its domain of application – as will be clear on the experimental results in next sections.

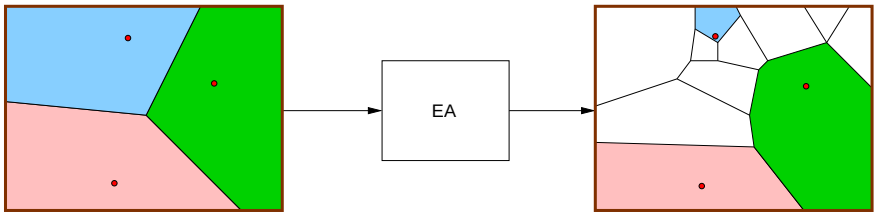


Figure 4.6: The effect of evolution on a set of *a priori* rules. The diagram on the left shows the partition defined by three *a priori* rules defined by their Voronoi sites. The diagram on the right show the application area of the same three *a priori* rules when more rules were added by the evolutionary algorithm (EA).

Figure 4.6 shows an example of the application areas of three *a priori* rules defined by their sites, and the application areas of the same rules at the end of an evolutionary process that add more rules. Note that the sites (center of the membership functions) are not modified by the evolutionary algorithm. The application area of the rules is modified indirectly when the evolutionary algorithm adds more rules in the system.

4.4 Control experiments

4.4.1 Cart-pole system

The cart-pole system or inverted pendulum system is a standard benchmark [115, 14, 98] used as an example of inherently unstable and dynamic systems. The problem consists (see figure 4.7) in determining the force to be applied, to push to the left or to the right, a cart with a pole on top of it. The pole must remain balanced and the cart must stay between the track boundaries.

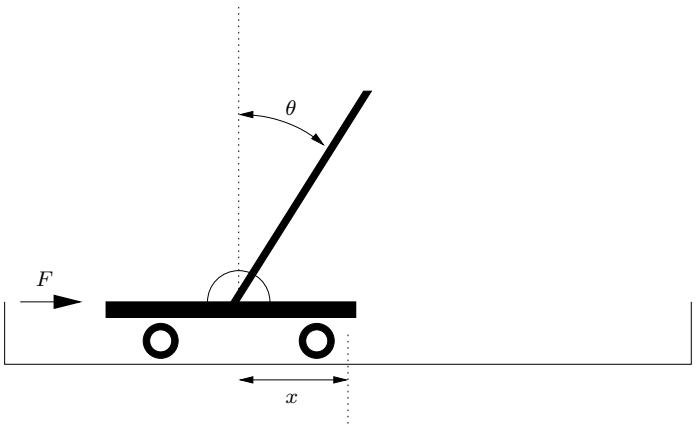


Figure 4.7: A schematic representation of the cart-pole system

The status of the system at time t is described by four variables:

$$\begin{aligned} \theta_t &= \textit{angle of the pole.} \\ \dot{\theta}_t &= \textit{angular velocity of the pole.} \\ x_t &= \textit{position of the cart.} \\ \dot{x}_t &= \textit{speed of the cart.} \end{aligned}$$

The motion equations that define the dynamics of the system in terms of the status variables and the force F_t applied at time t, are defined as follows:

$$\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t [\frac{-F_t - m_p l \dot{\theta}_t^2 \sin\theta_t}{m_c + m_p}]}{l[\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p}]} \tag{4.5}$$

$$\ddot{x}_t = \frac{F_t + m_p l [\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t]}{m_c + m_p} \tag{4.6}$$

where the same physical parameters used by [115, 14, 98] are used for the experiment:

$$\begin{aligned} m_c &= 1.0\ kg \textit{ (mass of the cart)} \\ m_p &= 0.1\ kg \textit{ (mass of the pole)} \\ l &= 0.5\ m \textit{ (distance from center of mass of pole to pivot)} \\ g &= 9.8\ m/s^2 \textit{ (acceleration due to gravity)} \end{aligned}$$

The system is simulated by approximating the motion equations using the Euler's method with a time step $\tau = 0.02$ seconds, and the following discrete time equations:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \tau\dot{\theta}_t \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \tau\ddot{\theta}_t \\ x_{t+1} &= x_t + \tau\dot{x}_t \\ \dot{x}_{t+1} &= \dot{x}_t + \tau\ddot{x}_t \end{aligned}$$

In the experiments, the cart-pole system is started from random positions by selecting random values for the input variables. Simulations are stopped when the pendulum falls down more than 12 degrees or the cart hits the boundaries located at 2.4 meters from the center. Note that there is no single solution for this control problem since any trajectory that do not pass through the regions of the space that have to be avoided is acceptable.

The results of experiments performed by considering respectively two input variables and four input variables are described below. In the first case, only the angle of the pole and its angular velocity are considered in order to compute the force. In this case, the objective of the control system is to keep the pole balanced and no checks are performed on the position of the cart. In the second set of experiments, both the angle and the position are considered.

Experiments in $\mathbb{R}^2$

This control problem consists in determining the force to be applied to the cart in order to keep the pole balanced by considering its angle and its angular velocity. The objective of the evolution is to obtain a Voronoi-based fuzzy controller that can control the cart pole system starting from random positions, keeping the pole in the accepted range, as close as possible to the equilibrium state.

The fitness assigned to the controller I is computed evaluating the global behavior of the simulated cart pole system when controlled by the controller I starting from r different positions (epochs) for a maximum of s time steps. In each epoch, the cart pole system is started in a random position, with the simulation stopped when the pole goes out of the limits, or the the maximum number of steps s is reached. Formally, the fitness is defined as the average of the fitness of the r runs as follows:

$$\textit{fitness}(I) = \frac{1}{r}\sum_{i=1}^{r} \textit{single}_i(I)$$

The fitness for a single run is defined as:

$$\textit{single}_i(I) = \begin{cases} \frac{|\theta|}{\theta_{max}} & \textit{succesful run} \\ \frac{s}{s-k_i}9+1 & \textit{non succesful run of length } k_i \end{cases}$$

where θ is the angle of the pole at the end of the simulation of a successful run, θ_{max} is the maximum allowed angle, and k_i the length of a non successful run. Note that the value of the fitness ranges from 10 (worst) to 1 (best) for non successful runs and from 1 (worst) to 0 (best) for successful runs.

Note that in most evolutionary studies where experiments with the cart pole system are performed, usually the fitness function is defined as the period that the angle of the pole (and the position of the cart) is (are) kept between predefined limits (see for example [115] and references there in). However, this definition of the fitness function, which is perfectly valid for theoretical studies, allows the evolutionary algorithm to obtain controllers that have an oscillating behavior, producing solutions that keeps the pole (and the cart) between the expected limits, but with the non desirable physical oscillation property. The fitness function defined here avoids this behavior by considering the value of the angle (and later on the position of the cart also) at the end of the simulated period. A possible drawback is that a wrong controller, that just by chance gets the pole (and the cart) at the right position at the end of the simulation period, can get a high fitness. However, since the total fitness for a controller is obtained by considering a number of runs started from random initial positions, the probability that a wrong controller will get the pole (and the cart) at the right position at the end of the simulated period in all runs is definitely rather low.

The fuzzy controllers for this experiment are defined with two inputs and one output. The two inputs correspond respectively to the angle and angular velocity of the pole, both normalized. The output corresponds to the force to be applied to the cart.

Experiments are performed without and with *a priori* knowledge. In the experiments with *a priori* knowledge, a single rule is incorporated as previous knowledge by default to all individuals that participate in the evolutionary process, and is not modified during the application of the variation operators. The rule is defined as follows:

site			output			
θ	$\dot{\theta}$	$\longrightarrow$	v	a_0	a_1	a_2
0	0		F	0	0	0

This *a priori* rule defines the behavior of the controller in the target position, establishing that the output produced by the controller (force) should be 0 when the angle and the angular velocity of the pole are both 0. Note that the behavior specified by the rule corresponds to the expected behavior in a single point of the domain and there is no mention on their application area.

The evolutionary algorithm for the evolution of the Voronoi based fuzzy controller individuals is run with the population size set to 50, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.6, with the perturbation relative weight set to 0.9, and relative weight for addition and removal of Voronoi sites set to 0.05 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 100. During the evolution, the evaluation is performed by computing the fitness as mentioned before,

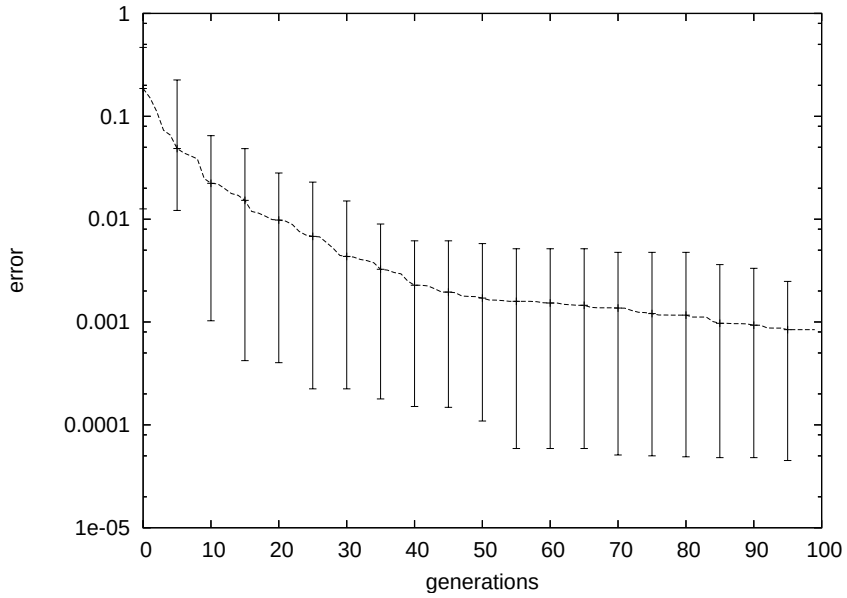


Figure 4.8: Last section of the best fitness plotted with error bars (in logarithmic scale) against the number of generations, averaged in 10 runs of the experiment without *a priori* knowledge.

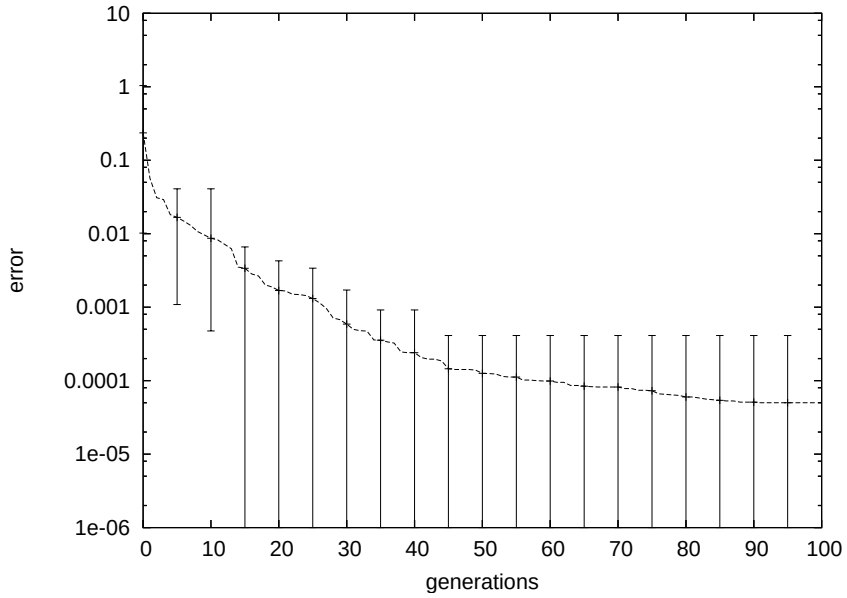


Figure 4.9: Last section of the best fitness plotted with error bars (in logarithmic scale) against the number of generations, averaged in 10 runs of the experiment with *a priori* knowledge.

with a maximum length of single runs set to 100, starting from 10 random positions. Each evolutionary run of the experiment took about 10 seconds execution time.

Figures 4.8 and 4.9 show the last section of the plot of the best fitness averaged in ten runs of the experiment, when the evolution is performed respectively without and with *a priori* knowledge. Note that the quality of the controllers is definitely better and standard deviation is much smaller when *a priori* knowledge is used.

Table 4.2 presents the results of the evolution of Voronoi-based fuzzy controllers with and without *a priori* knowledge. The columns labeled *fitness average*, *fitness best* and *fitness worst* correspond respectively to the average, best and worst fitness values obtained after the evolution averaged over 10 runs. It can be appreciated that the quality of the controllers is definitely better with *a priori* knowledge. The columns labeled *size best* and *size average* correspond to the size of the best individual and the average size of the population, both averaged over 10 runs.

a priori	fitness			size	
	average	best	worst	average	best
no	3.9569e-04	9.7096e-06	1.7308e-03	18.8	19.3
yes	2.4119e-11	1.9298e-12	7.3942e-11	17.6	17.4

Table 4.2: Summary of the results of the cart-pole system experiment in $\mathbb{R}^2$. The results presented in each row correspond to the average of the 10 runs.

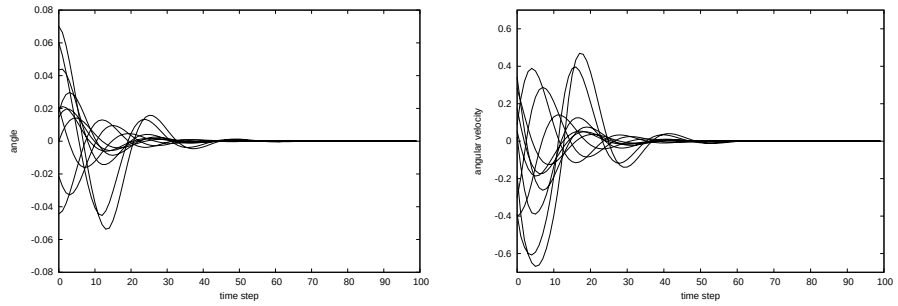
Figures 4.10 and 4.11 shows examples of execution of the best controller obtained in eight different runs of the evolutionary algorithm without and with *a priori* knowledge respectively. Diagrams on the left represent the change in the angle of the pole during the simulation and diagrams on the right the change in the value of the angular velocity. Each diagram summarizes 10 executions starting from random positions. Note that controllers evolved with *a priori* knowledge can reach the stable state much faster than controllers evolved without *a priori* knowledge.

Figure 4.12 shows the control function defined by a Voronoi-based fuzzy controller obtained through evolution. The base plane (which is rotated) defined by the Delaunay triangulation corresponds to the axis defined by the angle and the angular velocity of the pole. The value of the coordinate z corresponds to the force computed by the Voronoi-based fuzzy controller. The value at the center of the domain (point p_{17}) is zero (corresponding to the value defined by the *a priori* rule). The values are large positive in one side of the space (for big positive angles) and large negative in the opposite side.

It is interesting to note that the evolutionary algorithm can obtain successful controllers even in the case in which wrong *a priori* knowledge is inserted, showing the robustness of the proposed approach. The evolutionary algorithm inserts Voronoi sites near the wrong rule in such a way that its disrupting effect is reduced to a minimum. As an example, the same experiment is run with the following *a priori* rule:

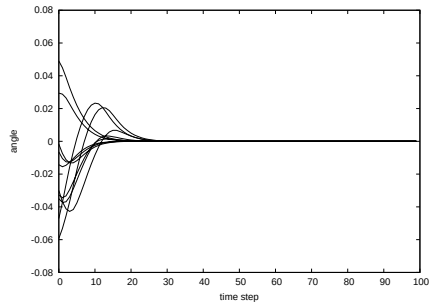
site		$\longrightarrow$	output			
θ	$\dot{\theta}$		v	a_0	a_1	a_2
0	0		F	1	0	0

This *a priori* rule defines that the output of the controller should be 1 when the angle and the angular velocity are both 0. The rule is wrong since in the rest position of the pole, makes the controller to push the cart to one side, throwing the system out of the target position.

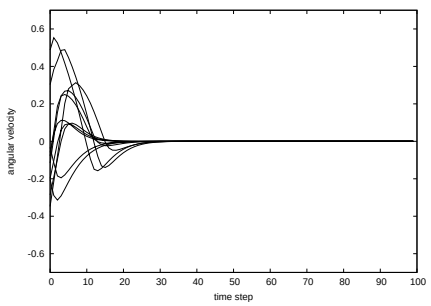


run 1: angle

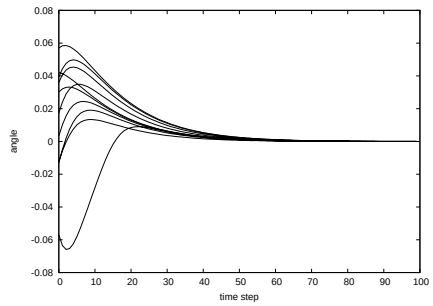
run 1: angular velocity



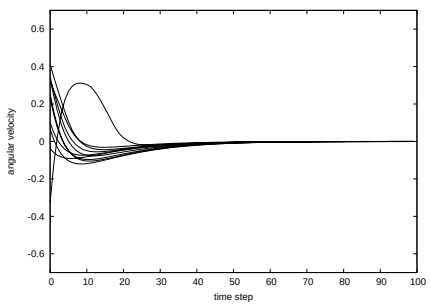
run 2: angle



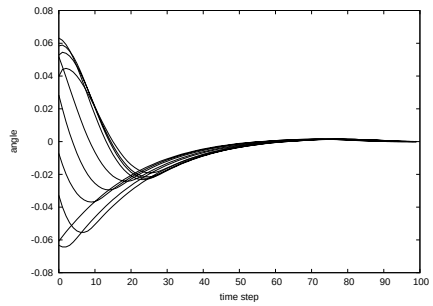
run 2: angular velocity



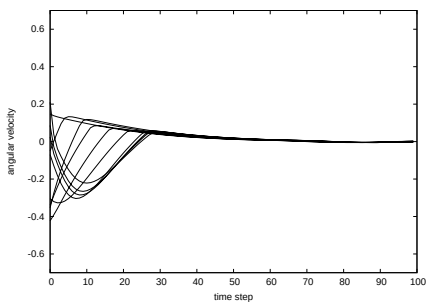
run 3: angle



run 3: angular velocity

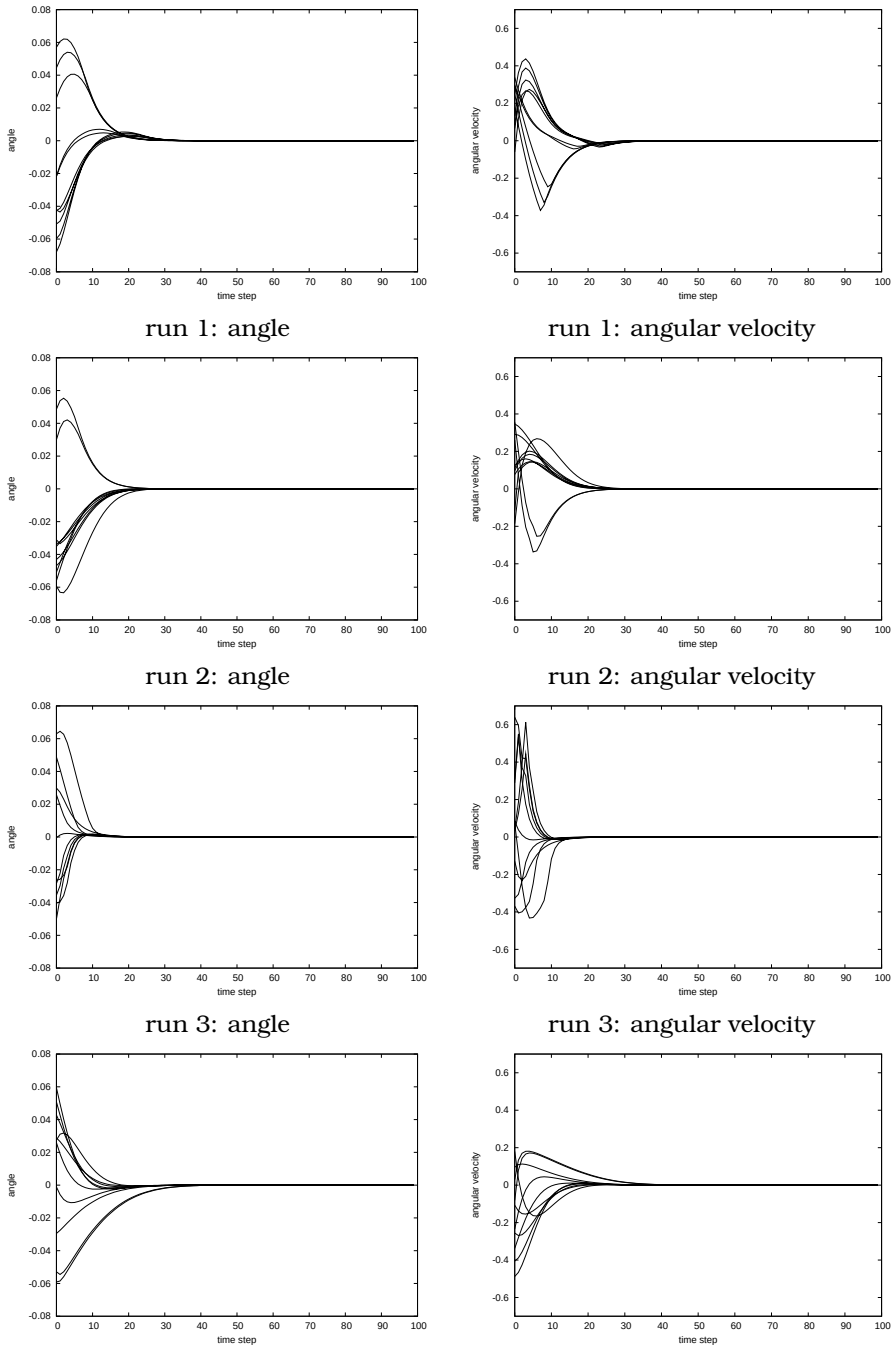


run 4: angle



run 4: angular velocity

Figure 4.10: Evaluation of the cart-pole system ($\mathbb{R}^2$) in 10 random experiments with the best controller obtained in 4 different runs of the evolutionary algorithm without *a priori* knowledge.



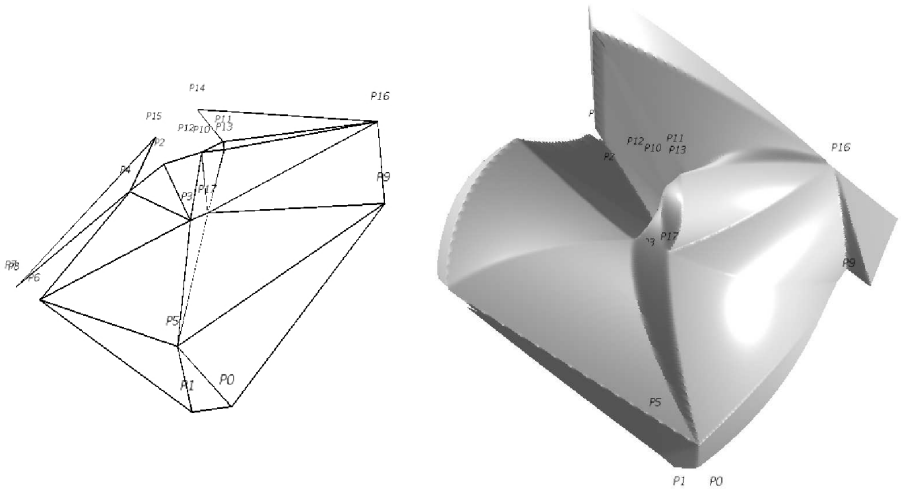
run 1: angle | run 1: angular velocity

run 2: angle | run 2: angular velocity

run 3: angle | run 3: angular velocity

run 4: angle | run 4: angular velocity

Figure 4.11: Evaluation of the cart-pole system ($\mathbb{R}^2$) in 10 random experiments with the best controller obtained in 4 different runs of the evolutionary algorithm with *a priori* knowledge.



(a) (b)

Figure 4.12: Example of the control function defined by a Voronoi-based fuzzy controller obtained by evolution. The plot on the left (a) shows the Delaunay triangulation and the plot on the right (b) the corresponding approximation. The value defined by the *a priori* rule is located in the center of the plots.

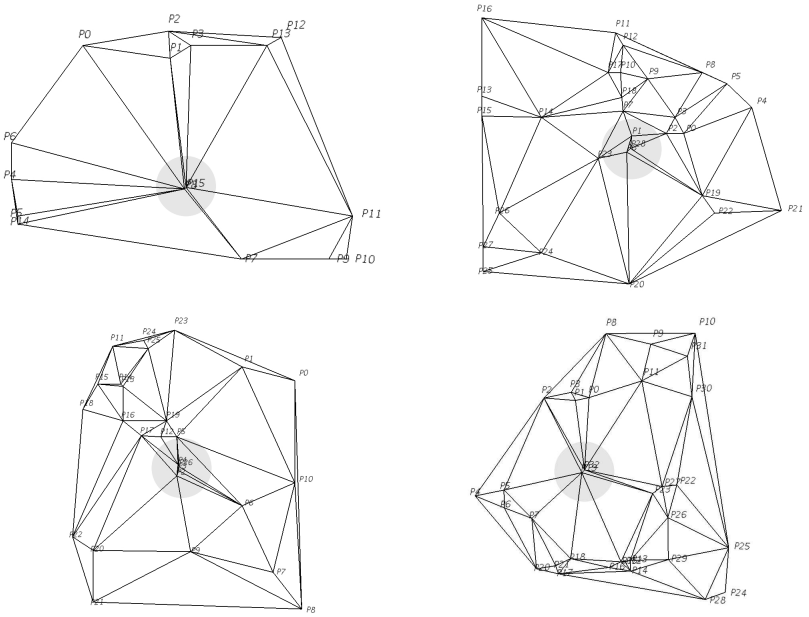


Figure 4.13: Delaunay triangulations of four approximations performed with wrong *a priori* knowledge. The dark circle is centered at the Voronoi site defined by the wrong rule

Figure 4.13 shows the Delaunay triangulations of four approximations generated by the evolutionary algorithm. The dark circle is positioned at coordinates (0,0), centered at the Voronoi site that defines the wrong rule. It can be seen that the algorithm positions other sites as near as possible to the wrong site in order to reduce the effect of its output. The average error obtained in ten runs is 0.0012, when the algorithm is executed with the same parameters used before, except the number of generations which is set to 500. It can be appreciated that the evolved controllers get an acceptable behavior, even if they include a wrong rule.

Experiments in $\mathbb{R}^4$

The problem consists in determining the force to be applied to the cart in order to keep the pole balanced and the cart between the limits of its track considering the angle and the angular velocity of the pole, and the position and the velocity of the cart. The objective of the evolution is to obtain a Voronoi-based fuzzy controller that can control the cart pole system starting from random positions, keeping the pole in the accepted range and the cart in the middle of the track, as close as possible to the equilibrium state.

The fitness assigned to the controller I is computed evaluating the global behavior of the simulated cart pole system when controlled by the controller I starting from r different positions (epochs) for a maximum of s time steps. In each epoch, the cart pole system is started in a random position, with the simulation stopped when the system goes out of the limits, or the the maximum number of steps s is reached. Formally, the fitness is defined as the average of the fitness of the r runs as follows:

$$\mathit{fitness}(I) = \frac{1}{r}\sum_{i=1}^{r} \mathit{single}_i(I)$$

The fitness for a single run is defined as:

$$\mathit{single}_i(I) = \begin{cases} \frac{|\theta|}{2\theta_{max}} + \frac{|x|}{2x_{max}} & \textit{succesful run} \\ \frac{s}{s-k_i}9 + 1 & \textit{non succesful run of length } k_i \end{cases}$$

where θ is the angle of the pole at the end of the simulation of a successful run, x the horizontal position of the cart, θ_{max} is the maximum allowed angle, x_{max} the maximum allowed value for x, and k_i the length of a non successful run. Note that the value of the fitness ranges from 10 (worst) to 1 (best) for non successful runs and from 1 (worst) to 0 (best) for successful runs.

The fuzzy controllers for this experiment are defined with four inputs and one output. The four inputs correspond respectively to the angle of the pole, its angular velocity, the position of the cart and its speed, all of them normalized. The output corresponds to the force to be applied to the cart.

Experiments are performed without and with *a priori* knowledge. In the experiments with *a priori* knowledge, a single rule is incorporated as previous knowledge by default to all individuals that participate in the evolutionary process, and is not modified during the application of the variation operators. The rule is defined as follows:

site					output					
θ	$\dot{\theta}$	x	$\dot{x}$	$\longrightarrow$	v	a_0	a_1	a_2	a_3	a_4
0	0	0	0		F	0	0	0	0	0

This *a priori* rule defines the behavior of the controller in the target position, establishing that the output produced by the controller (force) should be 0 when the angle and the angular velocity of the pole are both 0 and the cart is located in the center of the track with a speed of 0. Note that, as explained in the previous section, the behavior specified by the rule corresponds to the expected behavior in a single point of the domain and there is no mention on their application area.

The evolutionary algorithm for the evolution of the Voronoi based fuzzy controller individuals is run with the population size set to 50, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.6, with the perturbation relative weight set to 0.9, and relative weight for addition and removal of Voronoi sites set to 0.05 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 200. During the evolution, the evaluation is performed by computing the fitness as mentioned before, with a maximum length of single runs set to 500, starting from 10 random positions. Each evolutionary run of the experiment took about 5 minutes execution time.

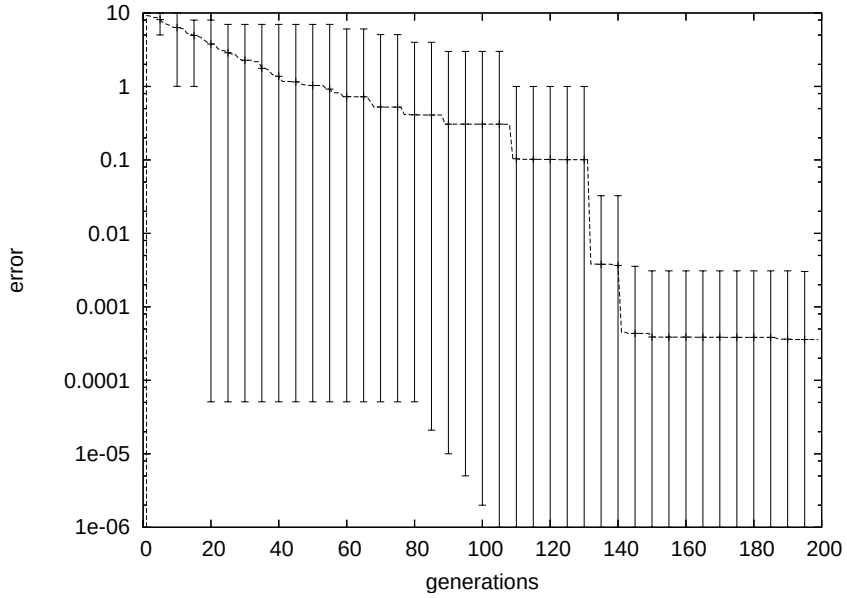


Figure 4.14: Last section of the best fitness plotted with error bars (in logarithmic scale) against the number of generations in 10 runs of the experiment without *a priori* knowledge.

Figures 4.14 and 4.15 show the last section of the plot of the best fitness averaged in ten runs of the experiment, when the evolution is performed respectively without and with *a priori* knowledge. Note that, like in the experiment of the previous section, the quality of the controllers is definitely better and the standard deviation is smaller when *a priori* knowledge is used.

Table 4.3 presents the results of the evolution of Voronoi-based fuzzy controllers with and without *a priori* knowledge. The columns labeled *fitness average*, *fitness best* and *fitness worst* correspond respectively to the average, best and worst fitness values obtained after the evolution averaged over 10 runs. It can be appreciated that the quality of the controllers is

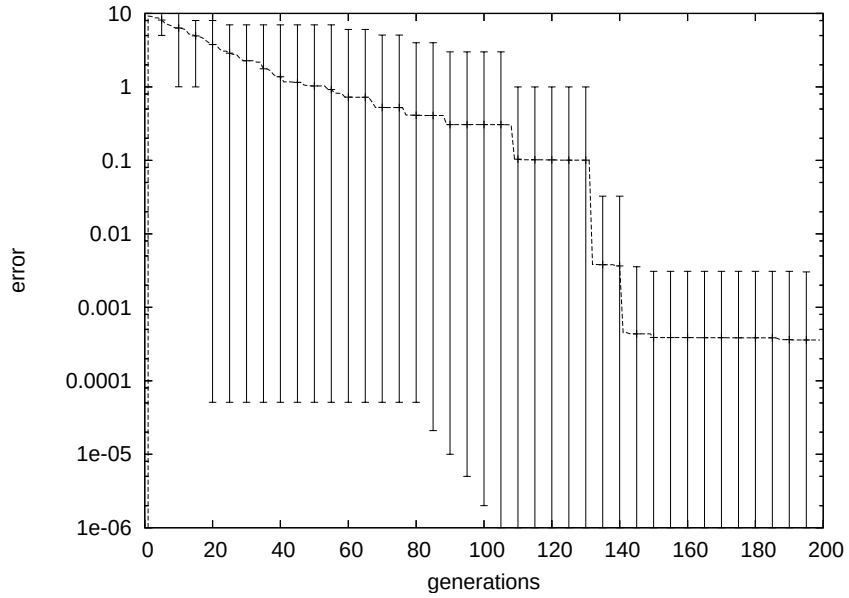


Figure 4.15: Last section of the best fitness plotted with error bars (in logarithmic scale) against the number of generations in 10 runs of the experiment with *a priori* knowledge.

better with *a priori* knowledge, however, there is not so large difference like in the previous set of experiments in $\mathbb{R}^2$. The columns labeled *size best* and *size average* correspond to the size of the best individual and the average size of the population, both averaged over 10 runs.

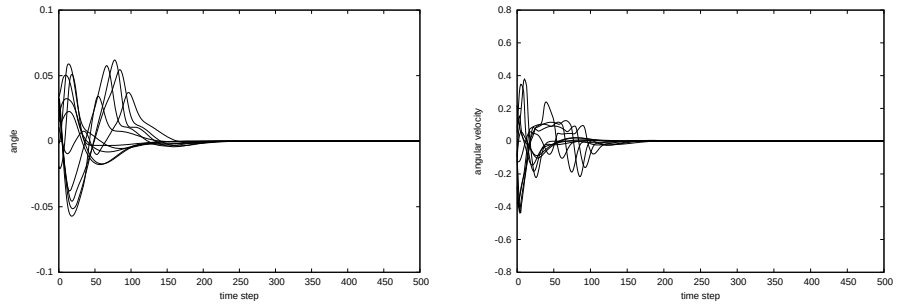
a priori	fitness			size	
	average	best	worst	average	best
no	1.3910e-04	2.5840e-07	4.2517e-04	14.0	15.0
yes	6.1049e-06	7.2496e-10	2.2464e-05	14.0	16.0

Table 4.3: Summary of the results of the experiment. The results presented in each row correspond to the average of the 10 runs.

Figures 4.10 and 4.11 shows examples of execution of the best controller obtained in four different runs of the evolutionary algorithm without and with *a priori* knowledge respectively. The diagrams represent the change in the angle and the angular velocity of the pole, and the change of the position and velocity of the cart during the simulation. Each diagram summarize 10 executions starting from random positions. Quality of the controllers is comparable.

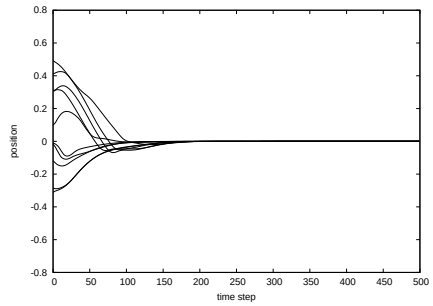
Discussion

The results show that the the evolutionary algorithm can obtain Voronoi-based fuzzy controllers for both instances of the cart-pole system problem. The experiments confirmed the intuitive hypothesis that *a priori* knowledge can enhance the quality of the resulting controllers. From the numeric results, it can be appreciated that the controllers obtained with *a*

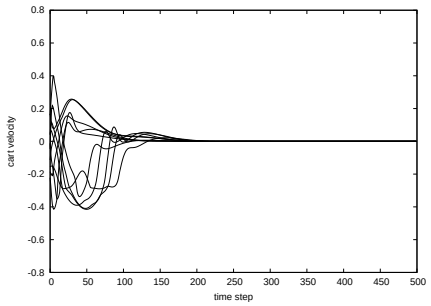


run 1: angle

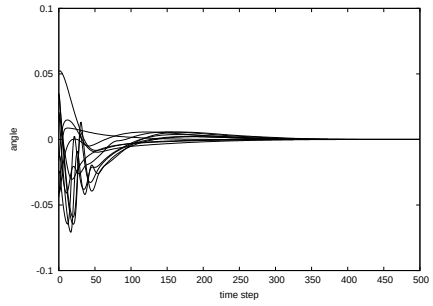
run 1: angular velocity



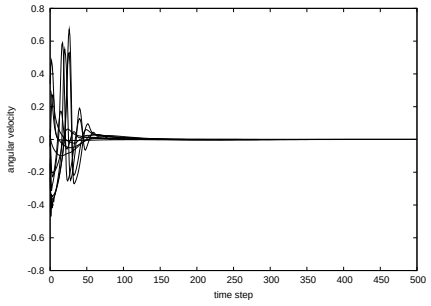
run 1: position



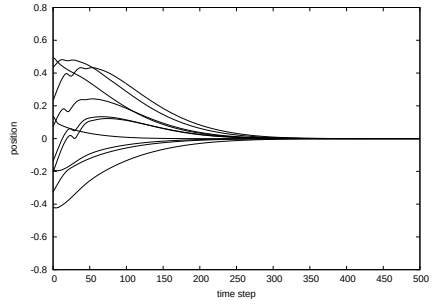
run 1: cart velocity



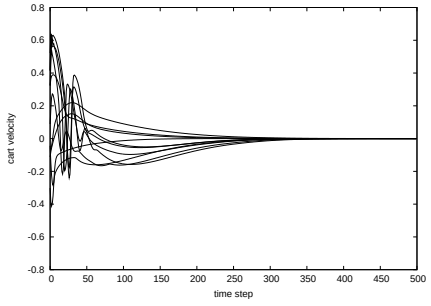
run 2: angle



run 2: angular velocity



run 2: position



run 2: cart velocity

Figure 4.16: Evaluation of the cart-pole system ($\mathbb{R}^4$) in 10 random experiments with the best controller obtained in 2 different runs of the evolutionary algorithm without *a priori* knowledge.

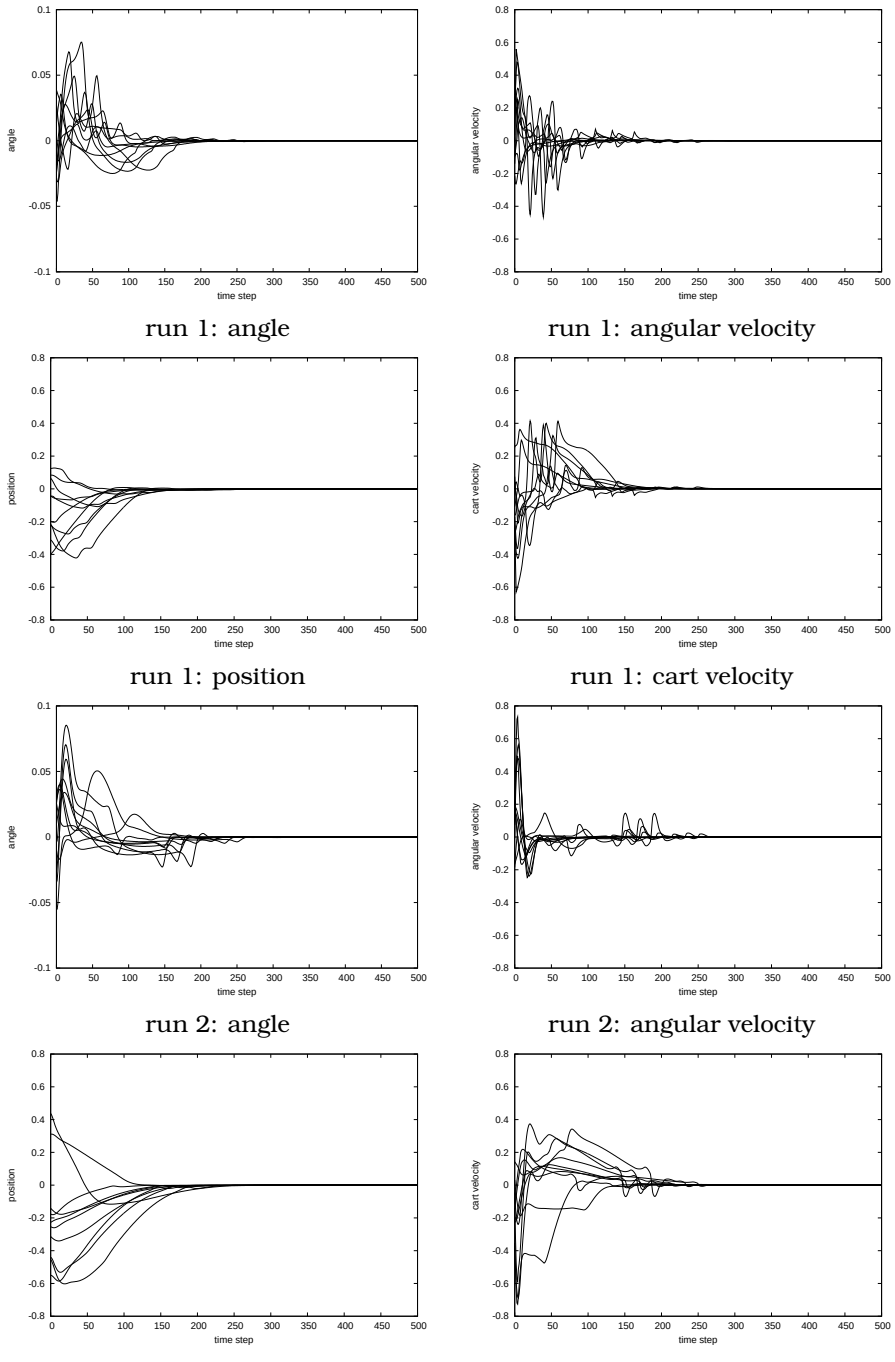


Figure 4.17: Evaluation of the cart-pole system ($\mathbb{R}^4$) in 10 random experiments with the best controller obtained in 2 different runs of the evolutionary algorithm with *a priori* knowledge.

priori knowledge are several orders of magnitude better in terms of the fitness function than the controllers obtained without *a priori* knowledge. This result is particularly significant by considering that just a single *a priori* rule was added in both experiments.

4.4.2 Evolutionary robotics

Evolutionary robotics aims at automatically developing controllers for autonomous robots through the use of evolutionary computation [121]. One of its main advantages is the fact that it releases the developer of the problem of deciding how to partition the desired behavior in simpler elementary problems. The role of the designer is limited to the specification of a fitness function, which is used to measure the robot ability in terms of its global behavior, with the hypothesis that complex behaviors arise from the interaction between the robot and the environment when driven by the resulting evolutionary pressure. However, the difficulty for the designer is to find, for each complex behavior he wants the robot to achieve, the correct action, or series of actions to be applied in turn, such that evolution does lead the controllers toward the desired behaviors.

Even if the evolution can be performed with a real robot, evolution with simulated robots is preferred since the serial evaluation of a single physical robot requires large amount of time. As an example, in an experiment performed by Floreano and Mandada [47] with real robots, one evolutionary run required 10 days. A typical solution consists in performing the evolution in a simulated environment and then evaluate the best individuals with the real robot. There is a risk that the controllers that perform correctly in the simulated environment do not reach the same quality level when evaluated in the real robot [24]. However, there is a large number of evolutionary experiments performed in simulated environments and then validated in real robots that show that, at least for a restricted class of problems, it is possible to obtain successful controllers with simulators, with almost identic behavior on real robots [121], due to the inclusion of noise that affects the simulation.

The Khepera robot

For its simplicity, because it has been the hardware basis for many experiments in evolutionary robotics, and because there exists many good simulation platforms, the Khepera robot was chosen.

The Khepera robot (see figure 4.18) is a miniature mobile robot with a diameter of 55 mm and a weight of 70 grams. It was developed by the Microprocessor Laboratory and Interfaces of the Swiss Federal Institute of Technology, Lausanne, Switzerland. It was specifically developed for this kind of experiments. From the point of view of its motion, it has two lateral wheels that can turn in both directions. The sensorial system consists in eight infrared sensors, located six on the front (two on the left side, two on the front and two on the right side) and two on the back. These sensors can be utilized in passive mode to measure the ambient light and in active mode to determine the distance to obstacles. The Khepera robot can operate in autonomous mode or under control of a computer connected with a high speed serial line, and recently with radio modules.

A large number of simulators have been developed in order to perform experiments without using a real Khepera robot. One of the simplest, but still very realistic is the *Freeware*

Figure 4.18: Khepera II robot

mobile robot simulator, developed by O. Michel from the University of Nice, France [111]. The simulated robot moves at the speed indicated by the controller. In order to obtain a more realistic simulation, the signal sent to both motors is perturbed with a random noise of $\pm 10\%$ of the speed amplitude. The direction resulting from the difference of speed between both motors is also perturbed with a random noise of $\pm 5\%$. Distance sensors are perturbed with a noise of $\pm 10\%$ of its amplitude and light sensors with $\pm 5\%$. The simulator allows the graphical definition of scenarios for robot operation and provides direct connection with a real robot if available.

Experimental settings

The selected problem to evaluate the Voronoi-based fuzzy systems generation is one of the most classical problems in evolutionary computation. The problem consists in the navigation through a scenario that has some obstacles. The robot has to travel the largest possible distance avoiding collisions with the obstacles. The objective of the evolution is to obtain a Voronoi-based fuzzy controller that can maximize the traveled distance of the mobile robot, carefully avoiding all obstacles during navigation.

The fitness assigned to a controller I is computed evaluating the global behavior of the simulated mobile robot when controlled by the controller I in a given scenario during a number of r epochs. In each epoch, the robot is started in a random position in the scenario, and fitness is accumulated at every step of the robot, proportionally to the speed, and reduced when the robot travels near obstacles, in order to favor navigation without collisions. The fitness accumulation is stopped when the robot bumps an obstacle, or it reaches a maximum number of steps s. Formally, the fitness is defined as follows:

$$\text{fitness}(I) = \frac{1}{rs} \sum_{i=1}^{r} \sum_{t=1}^{s} d(t) * (1 - a(t)) \tag{4.7}$$

where t is the time step, $d(t)$ is the normalized forward speed and $a(t)$ is the normalized maximum activation of the sensors [121]. The normalized speed $d(t)$ of the robot is defined as the summation of the forward speed of both motors in this time step. The value of $d(t)$ is 1 for a robot that is traveling at maximum speed in forward direction, and is 0 for a robot that is stopped or moving backwards. The value of $a(t)$ indicates if navigation is performed far away from obstacles ($a(t) = 0$) or very near them ($a(t) = 1$). This function assigns larger values

to individuals that travel at the highest speed, in a trajectory that follows (when possible) a straight line, and as far as possible to obstacles. The first term of the summation $d(t)$ varies between 0 and 1, with the objective to give higher fitness to individuals that travel at a higher forward speed. The term $1 - a(t)$ also varies between 0 and 1, with the objective to give higher fitness to individuals that are traveling away from obstacles. Note that the maximum value for fitness for the evaluation of an individual is 1, however, this value is theoretical, since in a real scenario it can not be obtained due to the presence of obstacles and paths that cannot be followed in a straight line.

The fuzzy controllers for this experiment are defined with four inputs and two outputs. The four inputs correspond respectively to the average of the two left sensors, the two front sensors, the two right sensors and the the back sensors. The two outputs correspond to the activation of the left and right motors. The reduction in the number of inputs by considering the average of the values of neighbor sensors is performed to simplify the controllers and their corresponding analysis. The figure 4.19 shows a diagram of the controller.

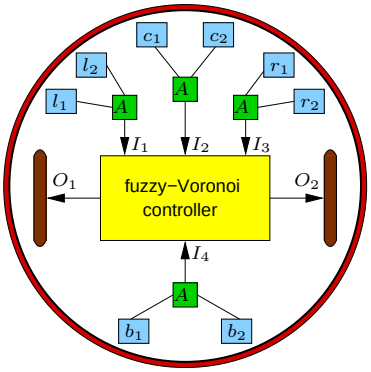


Figure 4.19: Diagram that represents the conceptual integration of the Fuzzy Voronoi controller in a Khepera robot. The inputs I_1, I_2, I_3 e I_4 correspond respectively to the average of pairs of input sensors. The outputs O_1 y O_2 provide directly the control signals to be applied to the motors.

Note that the fuzzy controllers that will be evolved in this problem, correspond to MIMO fuzzy systems, which can be considered as a set of MISO fuzzy systems, as it was discussed in section 2.4.3.

The behavior of the individuals (that represent fuzzy controllers) is evaluated in the scenario shown in figure 4.20, starting from multiple random initial positions.

The evolutionary algorithm for the evolution of the Voronoi based fuzzy controller individuals is run with the population size set to 50, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.6, with the perturbation relative weight set to 0.9, and relative weight for addition and removal of Voronoi sites set to 0.05 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 300. During the evolution, the evaluation is performed by computing the fitness as mentioned before, with a maximum length of single runs set to 200, performing 4 runs (epochs) from random initial positions. The number of rules of the initial controllers in the first generation is defined randomly in the range [10,30]. Experiments are performed without using *a priori* knowledge and

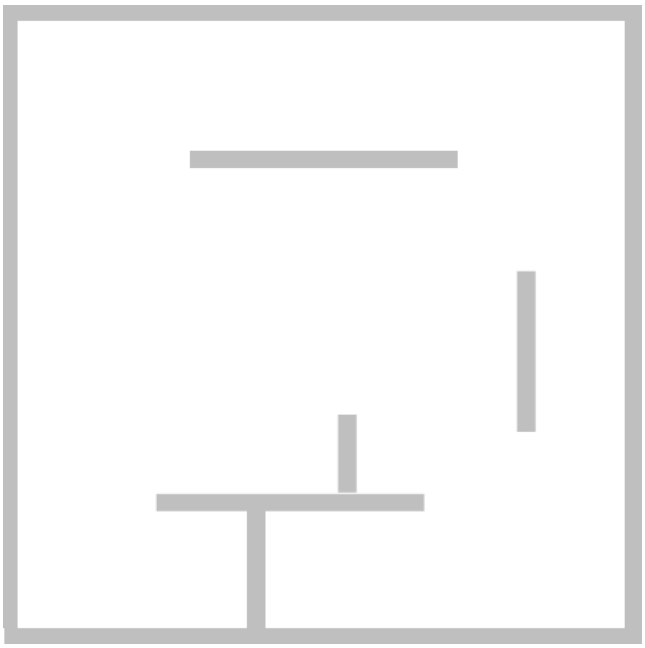


Figure 4.20: scenario used to evaluate the individuals of the population

then by incorporating two very simple rules. Each evolutionary run of the experiment took about 7 minutes execution time.

Experiments without *a priori* knowledge

Figure 4.21 shows the best fitness plotted against the number of generations, when evolution is performed without *a priori* knowledge. As noted before, a fitness value of 1 is not practically reachable due to the necessary limitations in speed and direction that the robot has to follow in order to navigate successfully in the scenario.

The performance of the best controller obtained after the execution of the evolutionary algorithm is shown in figure 4.22. The small square represent the position of the robot displayed every 30 time units, for a total of 500 time steps. It can be appreciated that the robot can successfully navigate through the scenario, avoiding obstacles, and moving slowly in narrow corridors (the effect can be appreciated by considering the density of the squares).

Figure 4.23 shows the relation between the terms $d(t)$ and $(1 - a(t))$ that define the fitness function (see equation 4.7) during the execution of the evolutionary algorithm for the best individual of each generation. The initial values corresponds to the low left value of the plot. It can be appreciated that both components of the fitness function are enhanced during evolution. In almost every generation, the evolutionary algorithm produces individuals that travel with larger forward speed and more far away from obstacles.

Figure 4.24 shows the change in the number of rules of the best individual in each generation. It can be noted that the number of rules is stable and does not grows indefinitely. This is a characteristic of the Voronoi crossover that has been verified in other contexts [128].

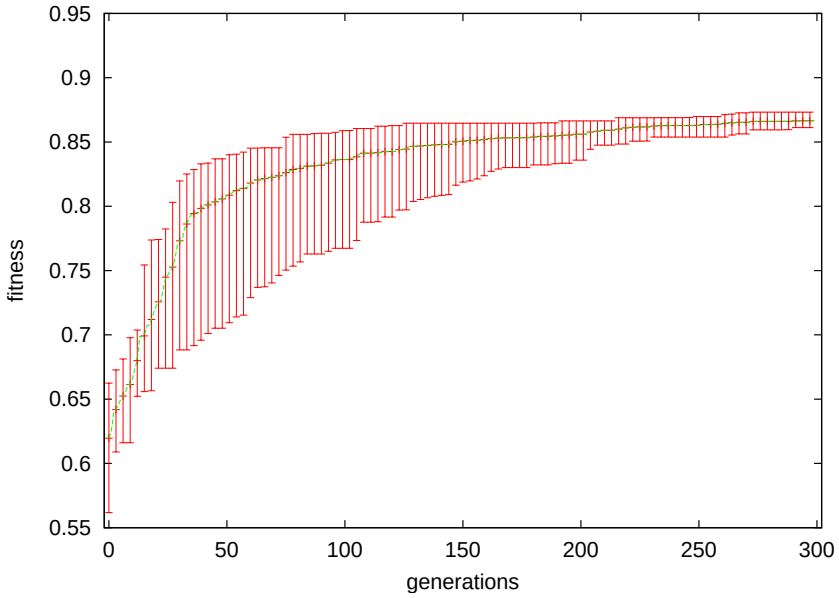


Figure 4.21: Best fitness plotted against the number of generations, averaged over 5 runs, together with error bars.

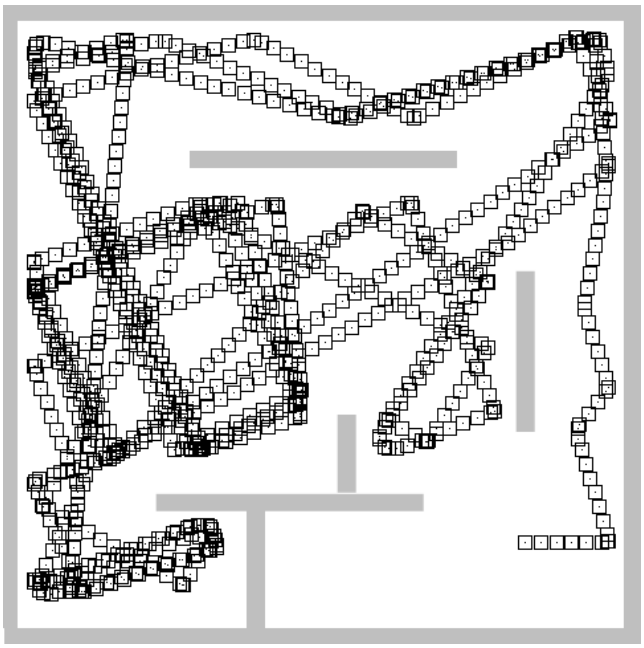


Figure 4.22: Performance of the best controller obtained after evolution without *a priori* knowledge.

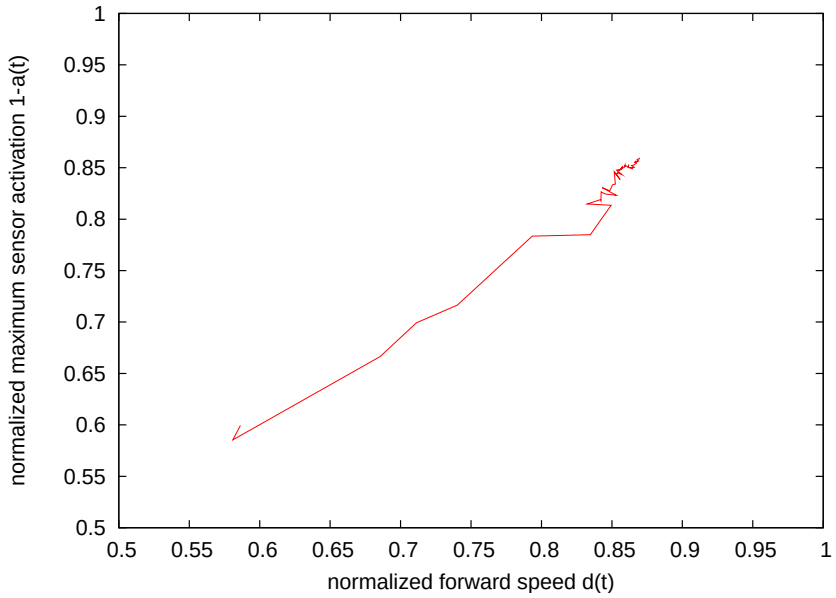


Figure 4.23: Relation between the terms $d(t)$ and $(1 - a(t))$ for the best individual along the generations of the evolutionary process.

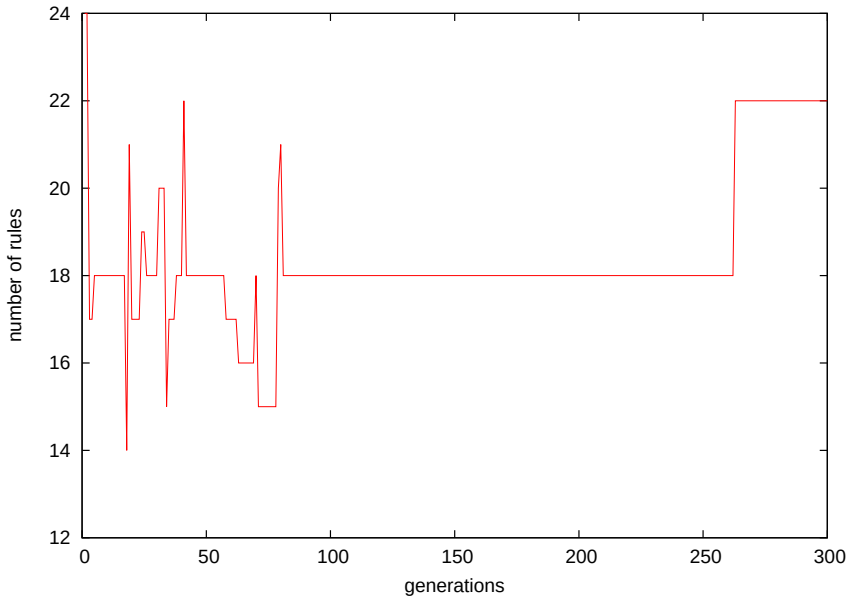


Figure 4.24: Number of rules of the best individual in each generation of the evolutionary process.

Figure 4.25 shows the performance of the best controller when evaluated on a scenario not used during the evolutionary process. The fuzzy controller is able to control correctly

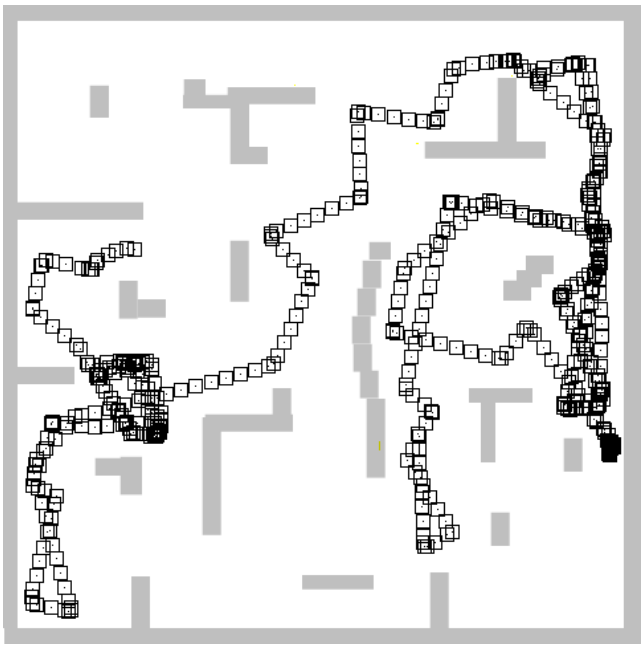


Figure 4.25: Performance of the best controller on a scenario not used during evolution.

the mobile robot, even in situation that are more complex that the ones considered during evolution. The evolutionary algorithm was able to obtain controllers that perform as expected independently of the particular scenario.

Experiments with *a priori* knowledge

The experiments were performed again, but this time using *a priori* knowledge. The two following rules were incorporated as the previous knowledge by default to all individuals that participate in the evolutionary process, which are not modified during the application of genetic operators:

site			
L	F	R	B
0	0	0	0

$\longrightarrow$

output					
v	a_0	a_1	a_2	a_3	a_4
v_0	1	0	0	0	0
v_1	1	0	0	0	0

site			
L	F	R	B
1	0	1	1

$\longrightarrow$

output					
v	a_0	a_1	a_2	a_3	a_4
v_0	1	0	0	0	0
v_1	1	0	0	0	0

The values that specify the points or sites defined by each rule (center of Voronoi regions) correspond respectively to the values of the average of the sensors located on the left (L), front (F), right (R) and back (B) of the Khepera robot. The values associated to the inputs are normalized in such a way that the value 0 indicates that no obstacles have been detected by

the sensor and the value 1 that there is a collision. The first rule establishes that the robot has to go forward at maximum seed (both motors at the highest speed, whose normalized value is 1) when there are no obstacles near by (normalized activation values of the distance sensors is 0). The second rule establishes the same behavior when there are no obstacles in the front (normalized value 0), but there are obstacles on the left and on the right (normalized value 1).

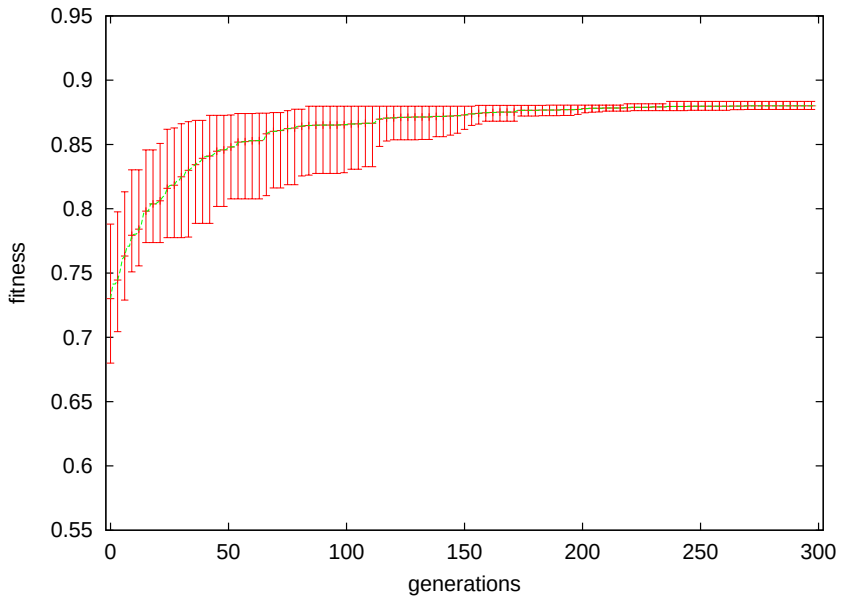


Figure 4.26: Best fitness plotted against the number of generations, averaged over 5 runs, together with error bars.

The figure 4.26 shows the best fitness plotted against the number of generations, when evolution is performed with *a priori* knowledge. Though the final performances reached by the best controllers in the case where *a priori* rules are used are only slightly better than those in the case without *a priori* rules, the results in the latter case are obtained more quickly, and, more importantly, are much more robust (smaller error bars). The use of *a priori* rules makes certainly sense when the simulation is highly costly so the number of runs and the number of generations per run have to be kept small.

The following rules are example of two rules obtained during evolution:

site			
L	F	R	B
0.61	0.98	0.97	0.92

$\longrightarrow$

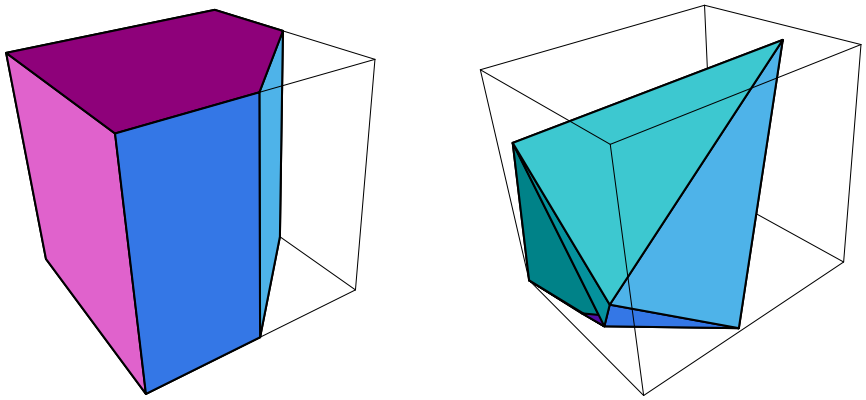
output					
v	a_0	a_1	a_2	a_3	a_4
v_0	-0.66	-0.51	-0.04	0.01	0.17
v_1	-0.82	0.9	-0.14	0.66	0.48

site			
L	F	R	B
0.82	0.54	0.11	0.67

$\longrightarrow$

output					
v	a_0	a_1	a_2	a_3	a_4
v_0	-0.12	0.65	-0.46	-0.11	3.31
v_1	-0.46	-0.14	-0.31	-0.11	3.54

The first rule applies when the only possible movement for the robot is to turn left and the second rule when the best option is a turn to the right. The normalized outputs produced by the first rule in its central point is 0.01 and 1.0, producing a fast turn to the left. The outputs of the second rule are 0.75 and 0.3 in its central point, producing a slow turn to the right.



(a) (b)

Figure 4.27: The (a) original area of application of the first *a priori* rule and the (b) modified area of application of the same rule at the end of the evolution. The graphics corresponds to projections of the four dimension areas of applications into the three dimension space by dropping the last coordinate.

The areas of application of the first *a priori* rule (projected in their first (three coordinates) are plotted on Figure 4.27: (a) shows the initial application area, before evolution started, while (b) is the application of the same rule after evolution. The algorithm did indeed adjust this domain of application – the user only had to specify the expected behavior at a single point in the input space.

Discussion

The experimental results show that, like in the cart-pole system problem, the evolutionary algorithm can obtain Voronoi-based fuzzy controllers that successfully solve the problem. The obtained controllers can guide the robot through the training scenario avoiding obstacles and traveling as fast as possible in straight line. But more importantly, the controllers have successfully learned the general navigation rules that drives a robot avoiding obstacles, as was shown when the controllers were evaluated in a new scenario, not used during the evolution. In both cases, the controllers get a comparable performance with the results provided in other experiments [121, 66], where neuro controllers are trained and evaluated in similar arenas.

Again, *a priori* knowledge demonstrated to be an important component of the evolution, since less resources (time and number of evaluations) are required to get controllers with comparable performance. As in the previous set of experiments, it is particularly interesting to note that just few rules (two in this case) are used to specify *a priori* knowledge.

The experiments show that the non standard approach provided by the Voronoi-based fuzzy systems, in which a rule is specified by just a single point and an action associated to it, allows (in some cases) easy interpretation of the rules generated by the evolutionary algorithm.

The same experiments were performed by considering controllers with eight inputs, corresponding to each one of the eight sensors of the robot (without averaging the values of neighbor sensors). In this case, slightly better quality results are obtained, showing that the evolutionary approach scales at least to this dimension. However, the extra time needed to run the algorithm (each evolutionary run took about 25 minutes) and the similar quality of the results, shows that the averaging approach is adequate for this problem.

4.5 Conclusions

This chapter has introduced a new approach to define fuzzy systems, based in the Voronoi approximation proposed in chapter 3. Two properties of the new model are particularly interesting: the synergic relation between the rules and the possibility to introduce easily defined *a priori* knowledge.

It is important to remark as mentioned before, that the use of membership functions defined in terms of the Delaunay triangulations was proposed by Benoit et al. [17]. However, it seems that no attempts were performed to exploit the synergic relations between the rules by using evolutionary algorithms.

The synergic relation between the rules makes possible to evolve fuzzy controllers by applying genetic operators that add, remove and modify rules by keeping fuzzy consistency in the individuals. In a standard fuzzy system, the rules are completely independent, meaning that changes in one of them do not affect the other rules. In a Voronoi-based fuzzy system, the area of application of one rule depends, not only on the rule itself, but also on the application areas of the neighbor rules. The area of application of the rules is immediately updated when a new rule is inserted, when a rule is deleted or when the application area of a rule is modified. This set of properties makes the proposed Voronoi-based fuzzy system a complete different approach in fuzzy system design, particularly well adapted for evolution.

Also, the possibility to insert *a priori* knowledge is very important. This property arises as a side effect of the symbiotic relation between the rules. In the proposed fuzzy system, a fuzzy rule is defined by just specifying the desired action in a single point. In this way, initial knowledge can be easily determined, since for most problems, it is not too difficult to determine which action should be performed in particular points on the input domain. This simple approach contrasts with the standard approach in fuzzy systems, were actions have to be specified for complete regions in the input domain, meaning that the relation between the actions has to be considered.

This powerful approach can be further extended to consider temporal problems, as next chapter shows.

Chapter 5

Recurrent Voronoi-based Fuzzy Systems

Even if most recurrent models proposed in the literature are successful in supporting learning of temporal sequences (see section 2.7), in most cases the semantic interpretation of recurrent units is not considered. This chapter introduces a recurrent structure for fuzzy systems defined in terms of the Voronoi-based fuzzy system. This new approach allows the definition of recurrent systems with a clear interpretation of recurrent units. The proposed recurrent structure consists in a set of rules, where the antecedents of the rules are determined by multidimensional membership functions defined in terms of Voronoi regions as in the standard Voronoi based fuzzy system. However, the difference is that the recurrent model includes internal variables with recurrent connections that allow the processing of temporal sequences of arbitrary length. Again, evolutionary algorithms are also proposed as the design tool.

This chapter is organized as follows. Section 5.1 describes the structure of the Voronoi based fuzzy recurrent model and section 5.2 discusses its properties. In section 5.3, the design of fuzzy recurrent systems with evolutionary algorithms is analyzed. Finally, section 5.4 presents experimental results on a system identification problem and an evolutionary robotic application.

5.1 Recurrent Voronoi-based fuzzy systems

This section presents an extension of the Voronoi-based fuzzy systems which includes recurrent connections [81]. The domain of application is extended to include temporal problems, where the output can depend on the current input and previous values of inputs and/or outputs. As it was described in section 2.7, this behavior is usually implemented by adding recurrent connections.

Definition 23 *Given a set of ω points $P = \{p_1, \ldots, p_\omega\}$, a MISO recurrent Voronoi-based fuzzy system is a Takagi-Sugeno recurrent fuzzy system with internal units, where the input space is partitioned in the ω regions defined by the Voronoi diagram induced by P and the fuzzy membership functions used in the antecedents of the ω rules correspond to the shape functions defined*

by the Voronoi approximation. Recurrent connections are allowed only on a set of internal units, which are not external inputs nor external outputs of the system.

A MISO [1] recurrent Voronoi-based fuzzy system in $\mathbb{R}^n$ has n external input variables, r internal input variables ($r \geq 1$), one external output variable v_1 and r internal output variables. The input vector $x =< x_1, \ldots, x_n, \ldots, x_{n+r} >$ of size $n+r$ is organized as the concatenation of the external input vector and the internal input vector. The output vector $v =< v_1, \ldots, v_2, \ldots, v_{r+1} >$ is organized as the concatenation of the external output and the internal output vector. The input vector x is fuzzified by ω fuzzy membership sets A_i ($1 \leq i \leq \omega$) whose multivariate membership functions μ_i ($i = 1, 2, \ldots, \omega$) are defined in terms of the Voronoi approximation. The output variables are defined as a linear combination of the input variables. The recurrent structure is implemented by defining the values of the internal input variables at time t as the value of the internal output variables at time $t-1$.

As an example, the definition of the k-th ($1 \leq k \leq \omega$) rule R_k follows:

$$\begin{array}{ll} \textbf{if} & x \textbf{ is } A_k \\ \textbf{then} & v_1 = a^1_{k0} + \sum_j a^1_{kj} x_j \\ & \ldots \\ & v_{r+1} = a^{r+1}_{k0} + \sum_j a^{r+1}_{kj} x_j \end{array} \tag{5.1}$$

where A_k is the multivariate antecedent fuzzy set associated to the input vector x, v_k is the value of the output variable v defined by the k-th rule, and a^j_{0k} and a^j_{ik} are adjustable real valued parameters ($1 \leq i \leq n, 1 \leq k \leq \omega, 1 \leq j \leq r+1$) associated to the j-th output variable.

It can be noted that there is a strong similarity here with the standard definition of Takagi Sugeno rules (see section 2.4.2). Except for the fact that now a single multidimensional set is used for membership, the main difference is that the fuzzy inference involves r terms in the input vector that are output values produced in the previous time step.

A schematic diagram of the recurrent Voronoi-based fuzzy system is shown in figure 5.1. The model is organized in four layers and consists of $n+r$ input variables, r internal variables, $r+1$ output variables and ω rules.

The first layer corresponds to the input layer, the second to the antecedent fuzzy memberships, the third to rules and the fourth to outputs. Units in layer one are input units. The first n input units are the external input units and the other units correspond to the internal inputs or recurrent units. Units in layer two are called partition units, acting as multidimensional fuzzy membership functions. Units in layer three are rule units, with each fuzzy rule in the fuzzy system having a corresponding rule unit. There is a one to one correspondence with units in layer two. Units in layer four are output units, being the first one the external output unit and the others the internal outputs or recurrent units. These units compute the outputs as a weighted linear combination of input units, generating both the external outputs and the values of the internal units to be made available as inputs in the next time step. The box labeled z^{-1} indicates one time step delay, meaning that the value of the internal input units at time step t correspond to the value of the internal output units at time $t-1$.

[1] As noted before, a MIMO fuzzy system can be considered as a set of MISO fuzzy systems (see section 2.4.3). Since that, all definitions in this section just consider MISO recurrent Voronoi-based fuzzy systems without loss of generality.

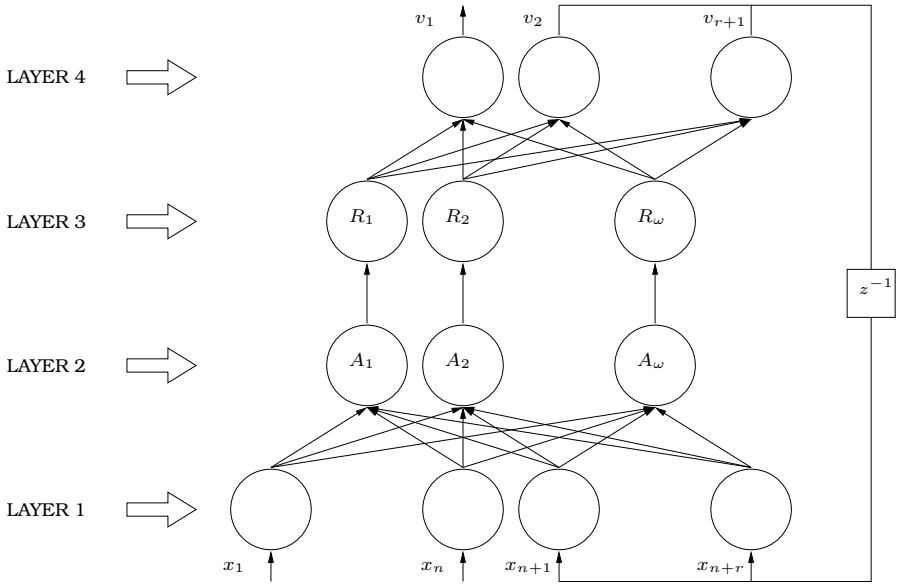


Figure 5.1: The structure of the recurrent Voronoi-based fuzzy system

The computation performed in the first three layers is identical to the computation performed by the non recurrent model described in section 4.1. Units in layer four produce the output vector v by computing the summation of the corresponding outputs as calculated by each rule. That is:

$$v_i = \sum_k v_i^k \tag{5.2}$$

Note that even if the internal inputs and outputs keep the same value at different time steps, they are different units. This fact allows the definition of fuzzy rules that consider them respectively as input and outputs to be used in the antecedents and consequents of the rules. However, since the number of input and output internal units in a recurrent fuzzy system is always the same, in order to simplify the presentation, fuzzy systems with r internal inputs and r internal outputs will be referenced simply as fuzzy systems with r internal units.

5.2 Properties

This representation provides the same set of properties of the Voronoi-based fuzzy systems (detailed in section 4.2), plus the ability to represent recurrent rules, with a clear semantic interpretation of the associated recurrent units.

5.2.1 Recurrent rules representation

The rules defined in the recurrent Voronoi-based fuzzy systems are standard Takagi Sugeno type fuzzy rules, with their own inputs and outputs. The complete system is recurrent because some outputs are connected to inputs, but each rule by itself is a standard Takagi Sugeno

type fuzzy rule, which contributes to provide a clear interpretation of the rules. This approach contrasts with other models like RFNN [94], RSONFIN [74], DFNN [102] or the TRFN [75], where the rules themselves include backward connections. The recurrent connection model is similar to the NFSLS approach proposed in [117], except that in NFSLS only a single output is considered and standard fuzzy partitioning is performed in the input domain.

5.2.2 Recurrent units with semantic interpretation

Recurrent units are defined as internal inputs and outputs of the recurrent Voronoi-based fuzzy system. In this way, internal units participate in the rule definition as any other input and output unit in the system. This fact contributes to provide a clear interpretation of the rules and makes it easy to define the recurrent *a priori* rules, as will be shown with examples in section 5.4.

5.3 Evolution of recurrent Voronoi-based fuzzy systems

The same representation and operators defined in section 4.3 are used to evolve recurrent Voronoi-based fuzzy systems. Note that even if recurrent connections have been added, there is no change in the structure of the fuzzy system (compare figures 4.1 and 5.1). Even the evaluation of a recurrent Voronoi-based fuzzy system is performed in the same way as defined by the algorithm 2.

However, the evaluation of a recurrent Voronoi-based fuzzy system performed in a temporal sequence has to pass back the values of internal outputs to internal inputs with one time step delay. An individual I is evaluated on a point p by using the algorithm detailed in pseudocode in listing 3.

```
1: procedure StepEvaluation(I, t, p)
2:     if t = 0 then
3:         x ← p + random
4:     else
5:         x ← p + s_{t-1}
6:     end if
7:     v ← Evaluation(I, x)
8:     s_t ←< v_2, ..., v_{r+1} >
9:     return v
10: end procedure
```

Algorithm 3: Pseudocode of the algorithm that computes the output produced by the recurrent Voronoi-based fuzzy system represented by the individual I when evaluated on the point p on time step t

The algorithm builds the complete input vector as a concatenation of the external input vector p with random values (line 3) for the first time step and with the previous values of the internal output units at time step $t-1$ in other case (line 5). The algorithm Evaluation is called with the complete input vector (line 7) in order to compute the outputs. The values of

the internal output units are stored in s_t in order to have it available for the next invocation at time $t+1$ (line 8).

5.4 Experiments

In this section, the evolutionary approach to design recurrent Voronoi-based fuzzy systems is evaluated on two problems. The first one is a system identification problem, where the outputs are defined in terms of past inputs and outputs. This problem is introduced in order to compare the approach with other methods. The second problem is an evolutionary robotic problem [121], where the ability to introduce *a priori* knowledge in the form of recursive rules is demonstrated.

5.4.1 System identification

The controlled plant is the same as used in the example 3 in [75] and is given by:

$$y_p(t+1) = \frac{y_p(t)y_p(t-1)(y_p(t)+2.5)}{1+y_p^2(t)+y_p^2(t-1)} + u(t) \tag{5.3}$$

where $y_p(t)$ and $u(t)$ are respectively the output and the input at time t. The desired output, which is graphically shown in figure 5.2, is defined by 250 pieces of data obtained from:

$$y_r(t+1) = \begin{cases} 0.6y_r(t) + 0.2y_r(t-1) + 0.6sin(2\pi t/45) & 1 \leq t \leq 110 \\ 0.6y_r(t) + 0.2y_r(t-1) + 0.2sin(2\pi t/25) & \\ \quad +0.4sin(\pi t/32) & 110 < t \leq 250 \end{cases} \tag{5.4}$$

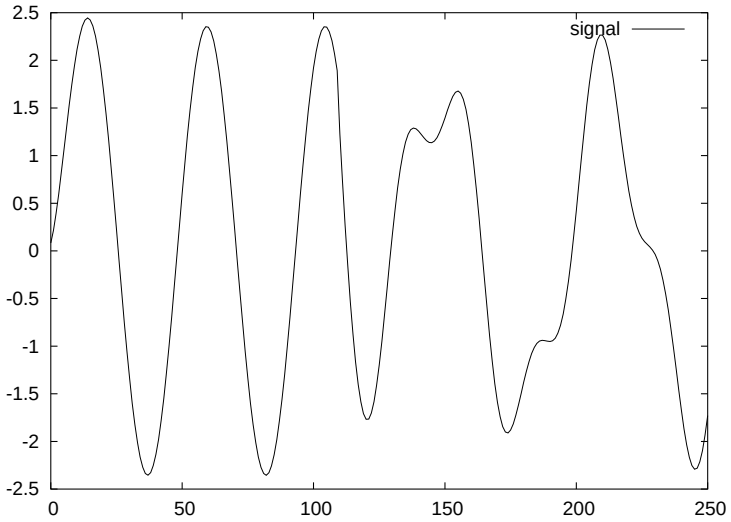


Figure 5.2: The control signal of the system identification problem.

Note that the value of the function y_r depends on the value of the function at previous time steps, causing that a non recurrent fuzzy system cannot learn the approximation.

The individuals are defined with one input, one output and one internal unit. The input is the value of the approximation at time $t-1$. The output corresponds to the value of the approximation at time t. Note that the presence of an internal variable (or internal unit) forces the rules to be defined with two inputs and two outputs (see figure 5.1). The evolutionary algorithm for the evolution of the recurrent Voronoi-based fuzzy individuals is run with the population size set to 50, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.5, with the perturbation relative weight set to 0.8, and relative weight for addition and removal of Voronoi sites set to 0.1 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 1200. The fitness is defined as the standard RMS error measured in the sequential approximation of the 250 points as follows:

$$\mathit{fitness}(I) = \sqrt{\frac{\sum_{t=1}^{250}(y_r(t) - \mathit{eval}_I(\mathit{eval}_I(t-1)))^2}{250}} \tag{5.5}$$

where $\mathit{eval}_I(\mathit{eval}_I(t-1))$ represents the evaluation of the recurrent Voronoi-based fuzzy system I by suplying as input the output of the system at time $t-1$.

Table 5.1: RMS error for the system identification experiment

RFNN+GA		TRFN+GA		RFV	
mean	best	mean	best	mean	best
0.3911	0.0850	0.0910	0.0536	0.0775	0.0235

The results for the best and average RMS error over 50 runs are listed in table 5.1. The table shows also the results obtained with the TRFN and RFNN models as presented in [75]. The comparison has to be considered with extreme care since, even if a careful selection of parameters was performed to replicate the experiments, there are some important differences in the models. For example, the Voronoi-based model uses variable length individuals, while the other methods use fixed length individuals. The main implication is that the number of fuzzy rules in the Voronoi-based model is determined by evolution, while in the other models has to be defined in advance. In the experiments with the TRFN and RFNN models performed in [75], the number of fuzzy rules is set to 4. The number of recurrent units is also 4, since in these models there is a one to one correspondence between the number of recurrent units and fuzzy rules. In the Voronoi-based model, the number of recurrent units (internal units) is not related with the number of rules, however, it has to be defined in advance (see section 5.1). and cannot change during evolution. For the current experiment, the number of internal units is set to 1. Each evolutionary run of the experiment took about 90 seconds execution time. Results in CPU time from [75] are not cited here since there is no information on the hardware used to run the experiments and the comparisons will not be fair.

Figure 5.3 shows the plot of best fitness (averaged over 50 runs) vs. the number of generations with the corresponding error bars. Due to the very small variation observed in the fitness value during the second half of the evolution process, only the first 500 generations are shown in the plot.

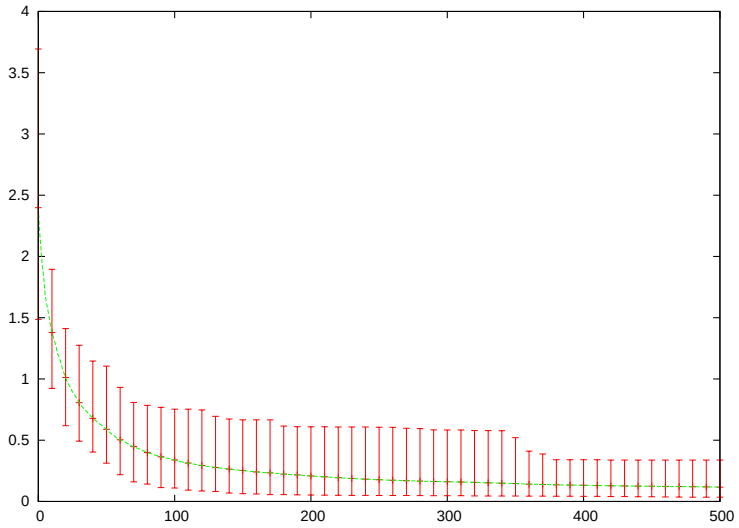


Figure 5.3: Fitness vs number of generations for the system identification problem with one internal unit, averaged over 50 runs.

The test signal used for evaluation is defined by the following equation:

$$\begin{aligned} y_r(t+1) = \quad & 0.6y_r(t) + 0.2y_r(t-1) + 0.2sin(2\pi t/25) \\ & +0.4sin(\pi t/32) \qquad\qquad 250 < t \leq 500 \end{aligned} \tag{5.6}$$

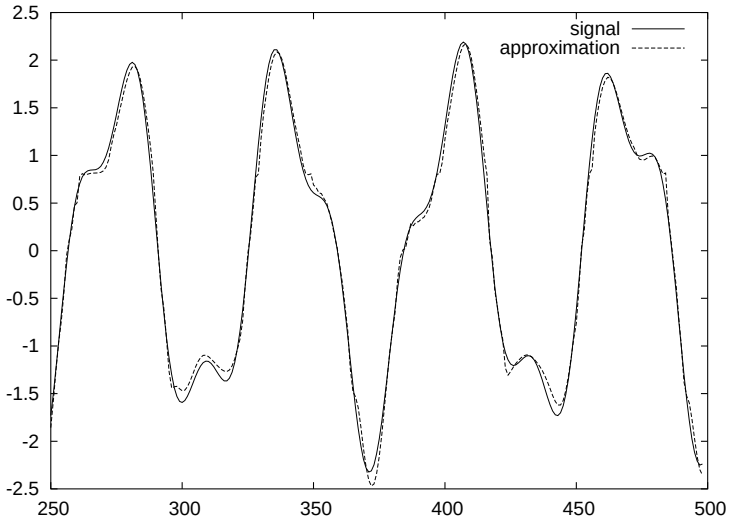


Figure 5.4: The best approximation in the system identification problem (dotted line) and the corresponding test signal (continuous line).

Figure 5.4 shows graphically the target control signal and the approximation of the best

recurrent fuzzy system.

Discussion

It can be appreciated that the evolutionary algorithm can obtain recurrent Voronoi-based fuzzy systems that provide a good approximation of the function, with results that are comparable in quality with the ones produced by other evolutionary approaches.

5.4.2 Evolutionary robotics

In this experiment, the objective of the evolution is to find a controller defined in terms of a recurrent Voronoi-based fuzzy system that can drive a Khepera robot avoiding collisions, starting from a fixed initial position, directed to a target position that depends on light based signals that are set to on or off status in the trajectory. The presence of an illuminated signal (*on* status) indicates to the robot that it has to turn left in the next intersection, and its absence (or *off* status) that it has to turn right. The controller needs internal memory, since the light signal is not present in the intersection, but in a previous (and may be distant) point in the trajectory. The controller has to learn also to *forget* light signals that affected the behavior in previous intersections and have not to be considered in other points of the trajectory.

The fitness of a controller is computed in a similar way as in [121], evaluating the controller in e different scenarios. Each scenario defines initial and target positions, and include path intersections where light signals determine the expected trajectory of the robot. The fitness is accumulated at every step of the robot proportionally to the speed, inversely proportional to the distance to the target point and reduced when the robot travels near obstacles, in order to favor navigation without collisions. The fitness accumulation is stopped when the robot bumps into an obstacle, or it reaches a maximum number of steps s. The total fitness is the average of the values obtained in the e scenarios. Formally, the fitness is defined as follows:

$$\text{fitness}(I) = \frac{1}{es} \sum_{i=1}^{e} \sum_{t=1}^{s} v(t) * (1 - a(t)) * (1 - d(t)) \tag{5.7}$$

where t is the time step, $v(t)$ is the normalized forward speed (summation of the speed of both motors), $a(t)$ is the normalized maximum activation of the sensors [121] (for example, $a(t) = 1$ implies a collision) and $d(t)$ is the normalized distance to the destination point (for example, $d(t) = 0$ implies that the target has been reached). This function assigns larger values to individuals that travel at the highest speed, in a trajectory that follows (when possible) a straight line, as far as possible to obstacles and minimizing the distance to the target point.

The controllers are defined with five inputs, two outputs and one internal variable. The inputs are, respectively, the average of the two left sensors (L), the two front sensors (F), the two right sensors (R), the two back sensors (B) and an average of ambient light (M) as measured by all sensors. The outputs correspond to the speed of the two motors. Note that the presence of an internal variable forces the rules to be defined with six inputs and three outputs (see figure 5.1) namely v_1, v_2 and v_3. The evolutionary algorithm for the evolution of the recurrent Voronoi based fuzzy individuals is run with the population size set to 50, the probability of Voronoi crossover set to 0.8, the mutation rate to 0.5, with the perturbation

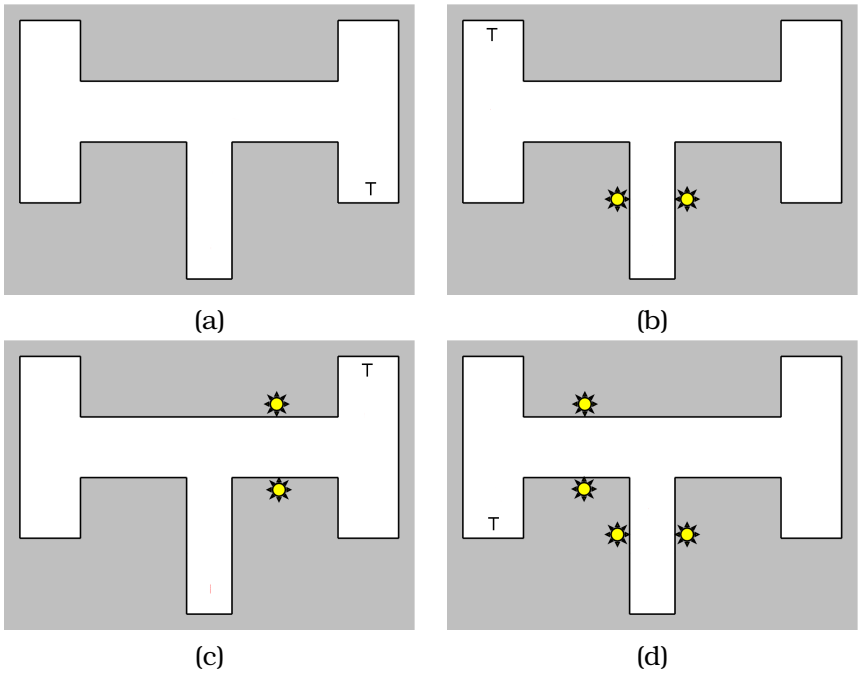


Figure 5.5: The four scenarios used to evaluate the performance of the controllers, where the target position is indicated by a **T** and the light sources with stars. In scenario (a) the robot has to run right in both intersections, in (b) the robot has to turn left first and then right, in (c) the robot has to turn right and then left and in (d) to the left in both intersections.

relative weight set to 0.8, and relative weight for addition and removal of Voronoi sites set to 0.1 each one. Selection is performed by tournament, no elitism is used and the maximum number of generations is set to 200. The performance of the individuals is measured in $e = 4$ scenarios with three intersections and all combinations of light signals, evaluated in at most $s = 500$ time steps. Figure 5.5 shows the four training scenarios. Each evolutionary run of the experiment took about 7 minutes execution time.

Rule	*site*	v_1	v_2	v_3
	L,F,R,B,G,M	$a_0^1,a_1^1,a_2^1,a_3^1,a_4^1,a_5^1,a_6^1$	$a_0^2,a_1^2,a_2^2,a_3^2,a_4^2,a_5^2,a_6^2$	$a_0^3,a_1^3,a_2^3,a_3^3,a_4^3,a_5^3,a_6^3$
R_1	0,0,0,0,0,0	1,0,0,0,0,0,0	1,0,0,0,0,0,0	0,0,0,0,0,0,0
R_2	0,0,0,0,0,1	1,0,0,0,0,0,0	1,0,0,0,0,0,0	1,0,0,0,0,0,0
R_3	0,0,0,0,1,0	1,0,0,0,0,0,0	1,0,0,0,0,0,0	1,0,0,0,0,0,0
R_4	0,0,0,0,1,1	1,0,0,0,0,0,0	1,0,0,0,0,0,0	1,0,0,0,0,0,0
R_5	0,1,0,0,0,0	1,0,0,0,0,0,0	0,0,0,0,0,0,0	0,0,0,0,0,0,0
R_6	0,1,0,0,0,1	0,0,0,0,0,0,0	1,0,0,0,0,0,0	0,0,0,0,0,0,0

Table 5.2: A priori rules. The value of the site corresponds to the center of the Voronoi region defined by the rule. It is specified by the normalized values of the left (L), center (F), right (R), back (B) and light (G) sensors, and the internal variable memory (M). The values a_j^i correspond to the parameters used to define the approximators.

The experiments were performed also by using *a priori* knowledge. The rules defined beforehand and inserted as explained in section 4.3.2 are shown in table 5.2.

The semantic of the rules is defined by considering the internal output variable v_3 (which corresponds to the internal input variable M, linked in the next time step with v_3) as a flag that indicates if a light signal was seen before. The first four rules correspond to the situation where there are no obstacles near the robot (all distance sensor values are equal to 0). The output produced in all cases for the motors is maximum forward speed (note the constant term of the approximator is 1 for both motor outputs v_1 and v_2). The value of the internal variable v_3 is set to 1 when light is present (rules 3 and 4) and to the previous value (can be 0 or 1) if no light is measured (rules 1 and 2). Rule 5 produces a turn to the right (left motor at maximum speed) if no light was detected before ($M = 0$) and rule 6 produces a turn to the left (right motor at maximum speed) if light was detected ($M = 1$). In both cases, the flag (internal variable M) is reset to 0. Again, it is important to note that the *a priori* knowledge is defined by specifying rules that determine the expected behavior of the controller in specific points in the input domain, without specifying the area of application of the rules, as it was detailed in section 4.3.2.

The results for the best and average fitness over 10 runs are listed in table 5.3, where the best possible value for fitness is 1 and 0 is the worst. Smaller error is achieved with *a priori* rules, but also the standard deviation is smaller, providing a more robust approach.

Table 5.3: Fitness for the evolutionary robot experiment

RFV			RFV + *a priori* knowledge		
mean	best	var	mean	best	var
0.7728	0.8723	0.0686	0.8510	0.8835	0.0255

Figures 5.6 to 5.8 show the values of the components of the fitness function (see equation 5.7) plotted for the best individual during each generation of the evolution in randomly selected evolution instances. The first component of the fitness is $v(t)$, which is the forward speed of the robot. The second component $(1 - a(t))$ can be considered as an index of safe navigation, with higher values when the robot navigates far away from obstacles. The third component $(1 - d(t))$ produces the maximum value 1 when the robot reaches the destination. The objective of the evolution is to maximize the fitness, maximizing the product of these three components in every step of the navigation. The initial values correspond to the low left value of each plot. The plots show that the three components of the fitness are enhanced during the evolution, sometimes one of them is enhanced at the expense of the others, but clearly, the evolutionary process is able to obtain individuals with better values in the three components when compared with the individuals in the initial population.

Figures 5.9 and 5.10 show the performance of the best controller obtained through evolution with and without *a priori* knowledge, when evaluated in the four test scenarios. Both controllers are able to drive robots that reach the correct destination when guided with the light signals. It can be appreciated that the controller defined with *a priori* knowledge performs faster turns in the intersections, which produce even a certain degree of instability in the direction of the robot. This effect is produced by the value selected in the *a priori* rules that establish maximum power for the motors in order to produce the turns and the unfortunate

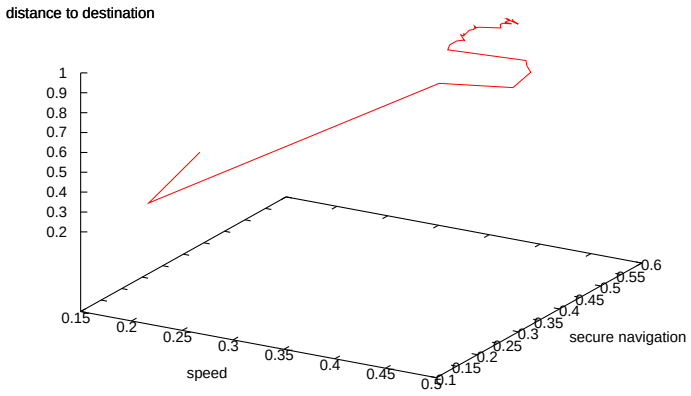


Figure 5.6: Relation between the terms $v(t)$, $(1 - a(t))$ and $(1 - d(t))$ for the best individual in each generation of the evolutionary process in one selected run of the evolutionary algorithm.

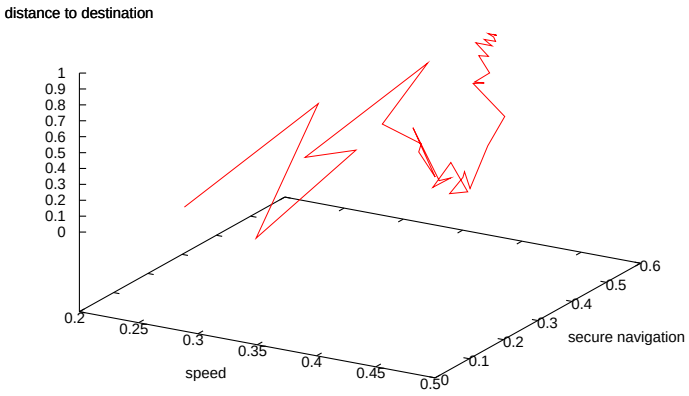


Figure 5.7: Relation between the terms $v(t)$, $(1 - a(t))$ and $(1 - d(t))$ for the best individual in each generation of the evolutionary process in other selected run of the evolutionary algorithm.

lack of intertia of the simulator. Effectively, the evolved fuzzy controllers perform as indicated by the *a priori* rules.

Figures 5.11 and 5.12 show the performance of the best controllers found during evolution in a scenario not used during evolution. The controllers are evaluated for 800 time steps. The controller that do not use *a priori* knowledge can drive the robot for a longer distance in the same number of time steps, performing not so abrupt turns in the intersections. Both

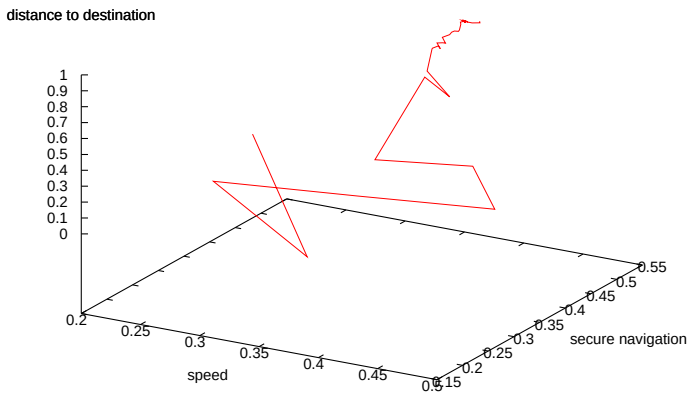


Figure 5.8: Relation between the terms $v(t)$, $(1 - a(t))$ and $(1 - d(t))$ for the best individual in each generation of the evolutionary process in still another selected run of the evolutionary algorithm.

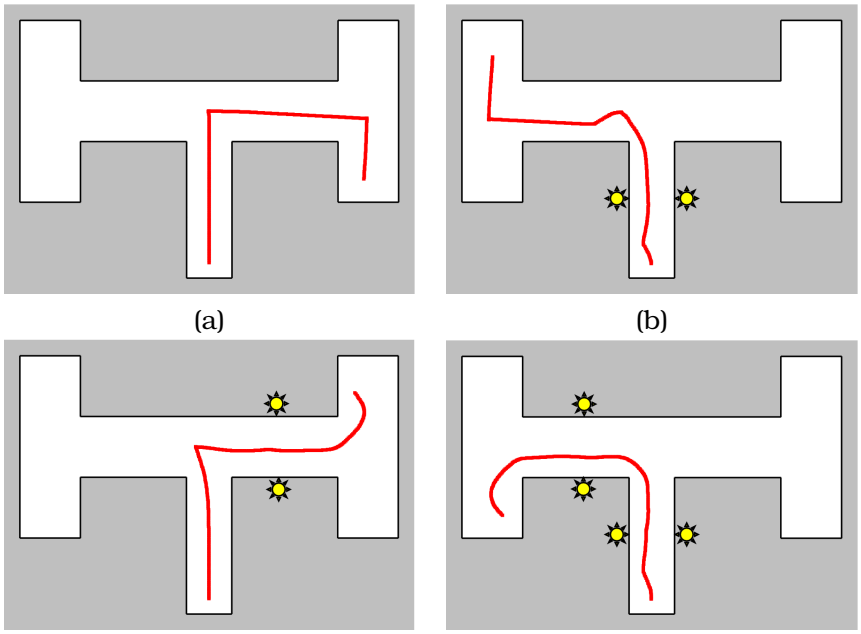


Figure 5.9: Performance of the best controller obtained through evolution without *a priori* knowledge.

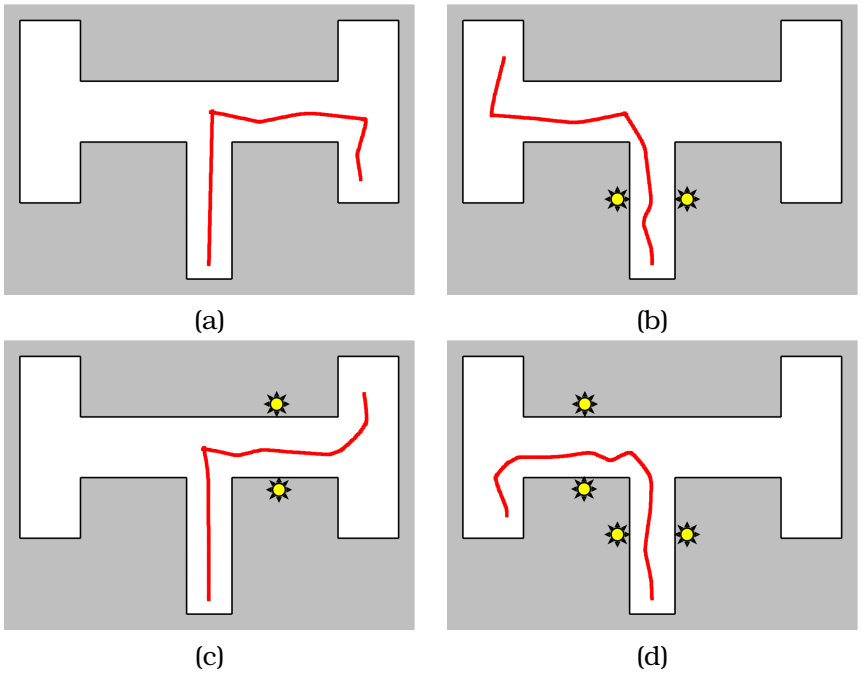


Figure 5.10: Performance of the best controller obtained through evolution with *a priori* knowledge.

controllers drive the robots by following the light signals as expected.

RFV			RFV + *a priori* knowledge		
mean	minimum	maximum	mean	minimum	maximum
20.9	11	35	26.5	14	38

Table 5.4: Number of rules of the controllers obtained through evolution, averaged over 10 runs.

The table 5.4 show the number of rules of the controllers obtained through evolution. The *a priori* rules do not participate in the evolution since they are only used during the evaluation of the individuals. It means that the real number of the rules for controllers evolved with *a priori* knowledge is the value shown in the table plus the number of *a priori* rules. From these results, it can be appreciated that the number of rules of a controller obtained without *a priori* knowledge is smaller. However, a very important point of the evolution with *a priori* rules, is that a definite semantic interpretation of the internal unit is provided: the internal unit value indicates if light was or not detected before the intersection.

Figure 5.13 shows the value of the internal unit of the recurrent controller evaluated on the four training scenarios corresponding to the execution of the same controller whose execution is shown in figure 5.9. It can be appreciated that the number of peaks in the plots correspond to the number of times light signals are detected in the path of the mobile robot.

Figures 5.14 and 5.15 show respectively the value of the internal unit of both best con-

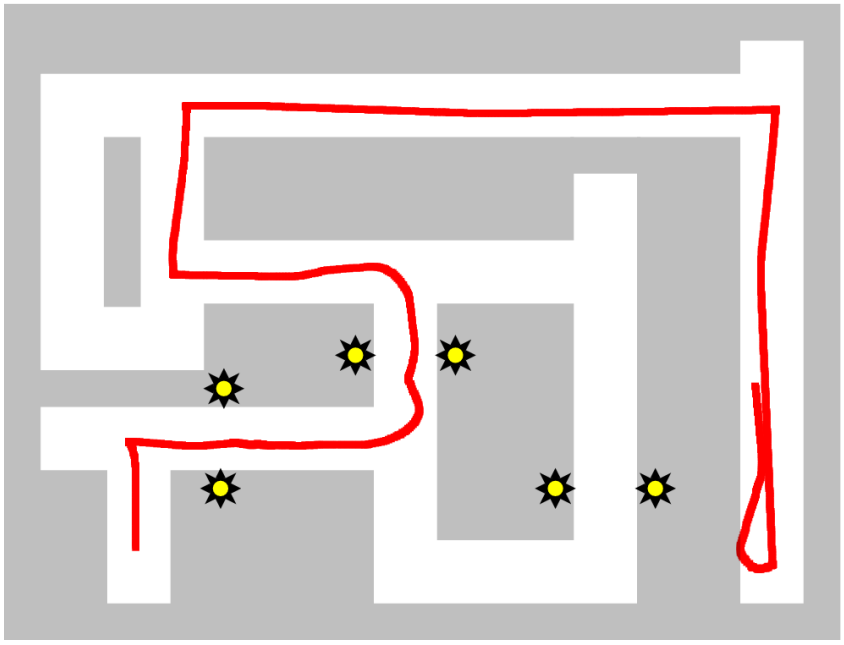


Figure 5.11: Performance of the best controller evolved without *a priori* knowledge when applied to the control of a robot in a scenario not used during evolution.

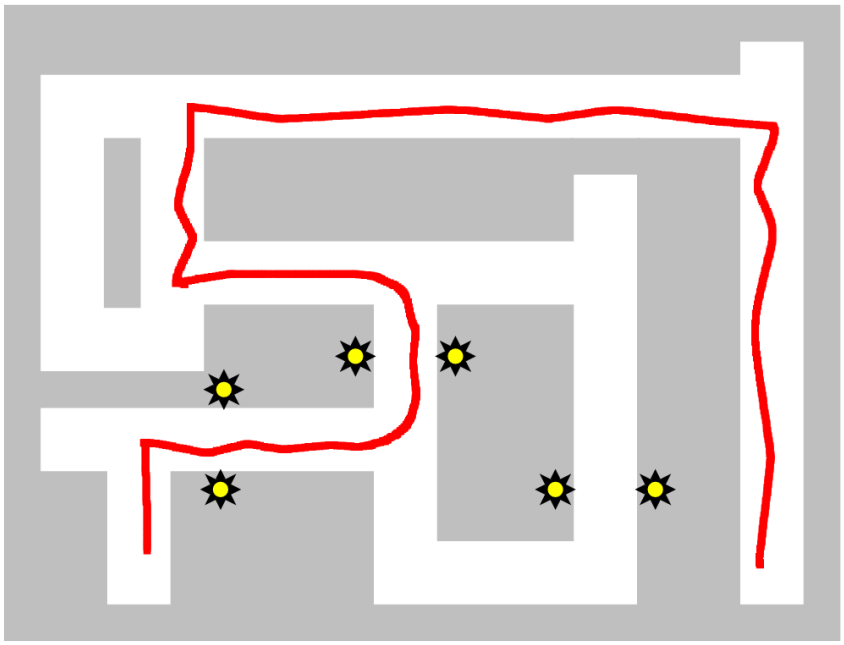


Figure 5.12: Performance of the best controller evolved with *a priori* knowledge when applied to the control of a robot in a scenario not used during evolution.

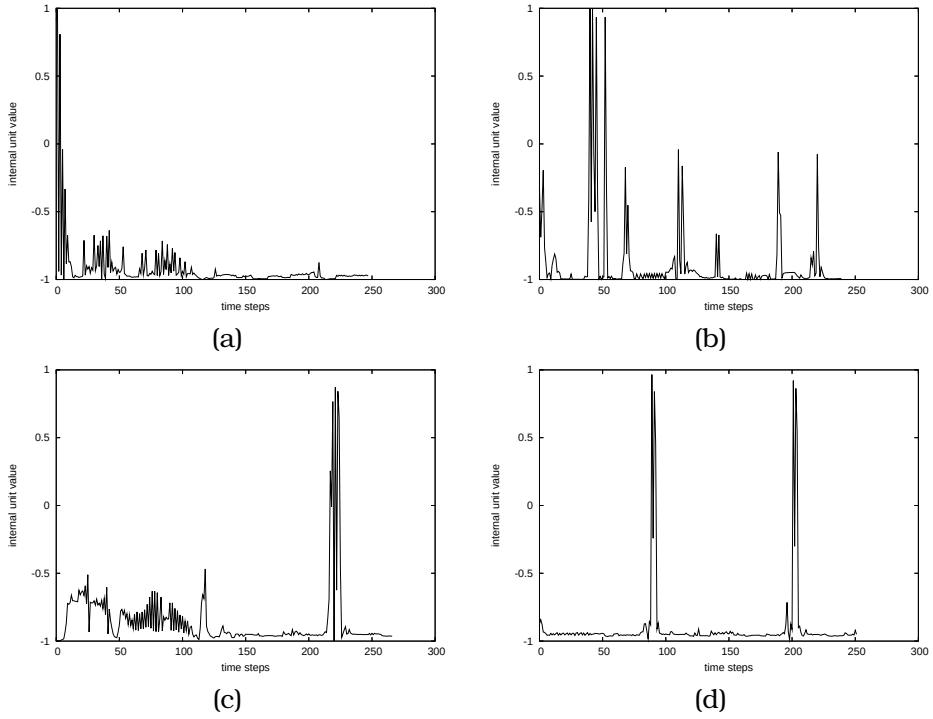


Figure 5.13: Value of the internal unit during the evaluation on the four test scenarios on selected runs of a controller obtained by evolution with *a priori* knowledge.

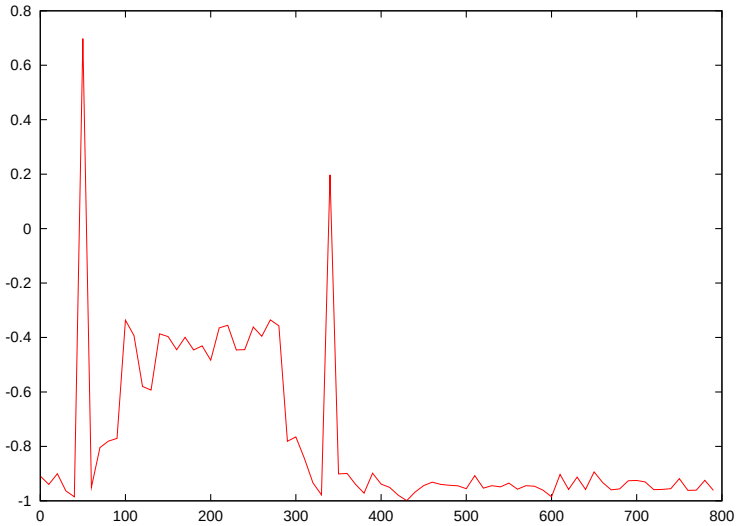


Figure 5.14: Value of the internal unit of the best controller obtained through evolution with *a priori* knowledge when evaluated on the test scenario.

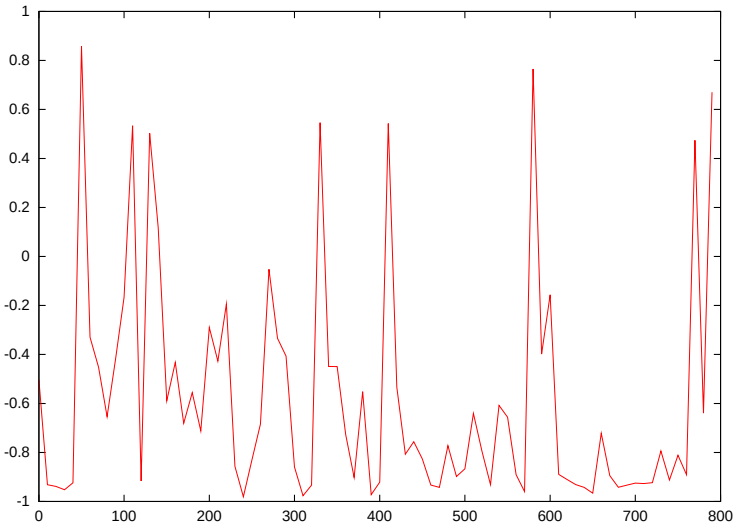


Figure 5.15: Value of the internal unit of the best controller obtained through evolution without *a priori* knowledge when evaluated on the test scenario.

trollers plotted for the 800 time steps when evaluated on the test scenario from figures 5.11 and 5.12 [1]. The value of the internal unit for the controller evolved with *a priori* knowledge represent the expected semantics, with two peaks on the areas where light signals were detected. No clear semantics can be defined in the case of the controller evolved without *a priori* knowledge.

Discussion

The evolutionary algorithm is able to obtain recurrent Voronoi-based fuzzy controllers that drive the simulated robot successfully in the evaluation scenario and also on a scenario not shown during evolution. The quality of the controllers is slightly better with *a priori* knowledge, with also smaller variance. However, the extra semantic that is provided to the internal unit is by far a very important achievement of the evolutionary process. The value of this unit can be used by other control process to monitor the status of the fuzzy controller or it can be even used as an input to other control system, since its semantic is well defined.

5.5 Conclusions

This chapter has introduced the recurrent Voronoi-based fuzzy system, an extension of the new fuzzy model proposed in chapter 4. The only difference between the recurrent extension and the non recurrent base model, is the existence of recurrent connections between output and input units. The inclusion of recurrent connections is typical for models that deal with temporal problems, as it was discussed in section 2.7. The recurrent model proposed in this

[1] Both plots were smoothed in order to enhance their readability

chapter follows the same approach, however, it provides a very interesting way to include *a priori* knowledge, a property inherited from the non recurrent Voronoi-based fuzzy system. In the extension proposed in this chapter, it is possible to include *a priori* knowledge that specifies complete recurrent rules, considering even the membership functions for the recurrent units. In most other fuzzy models it is not possible to specify completely the *a priori* recurrent fuzzy rules, since recurrency is introduced as external loops, something that cannot be specified in fuzzy terms. As an extra advantage, the model also provides an easy method for the definition of rules, since like in the base model, they are specified simply with a point and an action associated to it. Even if the rules are recurrent in this model, no extra complications are introduced for *a priori* knowledge definition, since the external input units and the recurrent units are treated in the same way, without making any difference between them.

Conclusions

Fuzzy systems are very powerful, since they can model nonlinear functions of arbitrary complexity. However, when compared with other approximation methods, their main strength is that they can be built based on the experience of experts. It is easier to define a system based on expert knowledge with fuzzy systems than with many other methods based in standard logic (or even with many other knowledge based approaches). The power of fuzzy systems comes from their ability to represent imprecise concepts and to establish relations between them. However, even if the method is very powerful, as demonstrated by the ever growing number of successful applications, it has some limitations. One of the main limitations comes from the dimensionality problem, which arises when the number of input variables is too high. It is important to stress that the problem is not due to the fuzzy system formal definition, which of course can deal with any number of input variables. The problem arises due to practical reasons, since with a highly dimensional problem, the knowledge provided by a human expert becomes difficult to extract and formalize. Usually it is not difficult to provide the definition of linguistic terms by considering one input variable at the time. However, the number of rules, which grows exponentially with the number of input variables in the classical definition of fuzzy systems, makes almost unmanageable the rule set definition process. Other problem concerns the quality of the obtained fuzzy controllers: in many cases, a fuzzy system with a reasonable performance can be easily defined based on expert knowledge, however, it can be still far from the optimum solution. For example, a typical problem in the definition of control systems is the unstable behavior in some areas of the input space. The main reason for this problem is usually the difficulty to define optimum shapes for the membership functions. Neural networks and evolutionary algorithms are standard approaches that can help in the optimization of fuzzy systems (see chapter 1).

One of the main contributions of this thesis is the proposal of a new type of fuzzy system: the Voronoi-based fuzzy system (detailed in chapter 4). This model, which belongs to the Takagi-Sugeno class, presents two main properties: (1) the fuzzy membership functions are multidimensional and (2) the rules are organized in a synergically related set. The antecedents of the fuzzy rules are defined by single points in the input domain, and based on them, a set of multidimensional linguistic labels is defined implicitly, fulfilling the usual requirements for fuzzy systems (see chapter 2). The possibility to define knowledge in such a way, is particularly useful in control problems, where it is very easy to define the expected output of the fuzzy system (controller) in some specific points of the input space, while it is very difficult to determine the general control rules. As mentioned before, and as a very unusual property for fuzzy systems, the rules in the Voronoi-based model are not independent, and changes

in the area of application of one rule affects the area of application of neighbor rules. These two properties of the proposed model makes it a good candidate for evolution, since (1) a very simple representation can be defined because the antecedents of the rules are defined just with single points, and (2) the variation operations can profit from the synergism of the rules.

The proposed fuzzy model has a strong theoretical support, since it is defined formally in terms of the Voronoi approximation proposed in chapter 3. The Voronoi approximation is a method that performs function approximation based on geometric concepts: Voronoi diagrams, Delaunay triangulations and barycentric coordinates. The Voronoi approximation belongs to the family of local approximation methods, where the complete approximation is built as a combination of a set of local functions. Each local function is applied in a definite region of the space, which in our model is defined by the Voronoi diagram induced by a set of points. The Voronoi approximation is built by using an evolutionary algorithm, which evolves both (1) the partition of the input space and (2) the set of local functions. As an interesting property, it was shown experimentally that the evolutionary algorithm assigns more local approximators in the areas that are more difficult to approximate, and less local approximators in easier areas of the domain. In this sense, by assigning a proper number of local approximators, the method is more efficient in terms of resources than the standard approach used in most mesh-based finite elements methods.

The Voronoi-based fuzzy system is just a Voronoi approximation, with a different semantic associated to the component elements (see table 4.1 for a comparison). Since the two models are two different views of the same base basic structure, all results obtained with the Voronoi approximation are directly applicable to the Voronoi-based fuzzy system. In fact, the representation used for evolution was proposed for the approximation, and then used directly for the evolution of fuzzy systems.

The ideas proposed can be further extended to cover other domains of applications, and also, different approaches can be considered to potentially enhance the quality of the approximations. Interestingly, due to the duality of the model, extensions of the Voronoi-based fuzzy system can be defined by considering ideas coming from both the formal approximation theory and from the fuzzy logic area.

An example of an extension that was proposed in this thesis is the definition of Voronoi-based recurrent fuzzy systems, approach that was introduced in chapter 5. In a recurrent Voronoi-based fuzzy system, a set of recurrent connections was added between internal outputs and internal input units. In this way, the area of application of the model was extended to consider also temporal problems, while still maintaining the same properties of the original model. In the recurrent extension, *a priori* knowledge can be added for the evolution as a set of recurrent fuzzy rules. In these fuzzy rules, the membership functions of the internal recurrent units can also be specified, something that is unusual for most models of recurrent fuzzy systems. In this way, it is possible to associate a semantic to the recurrent internal units by a careful selection of the *a priori* knowledge. Since the antecedents of the fuzzy rules are specified with a single point, which includes the values of recurrent units, the specification of recurrent fuzzy rules is as simple as the specification of non recurrent rules.

Other areas for extensions can arise by considering the different classes of fuzzy system models. The proposed fuzzy system belongs to the class of Takagi-Sugeno models, however, it is possible to consider the definition of a Mamdani Voronoi-based fuzzy system, just by changing the definition of the consequents of the rules. Instead of using local linear approx-

imators, it is possible to define a partition of the output space in terms of fuzzy sets, and associate the rules with the linguistic terms in which the output is partitioned. Note that the main properties of the Voronoi-based fuzzy system (the partition of the input space with multidimensional membership functions and the synergic rule set) are not affected at all and the fuzzy system maintains the set of advantages discussed before. This idea can also be pushed further: the partition of the output space can be done in terms of Voronoi regions. From the implementation point of view, a set of points in the output domain has to be evolved together with the individuals, and the consequent of the rules has to be associated with them. The set of consequent points does not need to have the same size of the set of rules, allowing rules to share consequent membership functions, as usual in Mamdani type fuzzy systems. The output of the complete fuzzy system is then computed by defuzzificating the output values as indicated by the multidimensional output fuzzy sets induced by these points. Again, the duality of the models helps at this point, since the defuzzification of the multidimensional output fuzzy sets can also be defined in geometrical terms.

It is also possible to propose more extensions by considering the basic Voronoi approximation. In the original model, the shape functions are defined as linear functions in the input space, where the value is 1 in the center of the region, and goes down to 0 in the center of the neighbor regions. An extension can consider the use of nonlinear functions, for example, bell-shaped functions that fulfills the same condition mentioned before. The system still corresponds to a fuzzy system, where the multidimensional membership functions are now continuous and differentiable. The same approach for evolution can be used, for both the Voronoi approximation and the Voronoi-based fuzzy system, however, other optimization techniques can now be applied. Since all functions are now differentiable, algorithms based on gradient descent methods, like neural networks can be applied. This new definition implies that it could be possible to evolve a fuzzy system, including also *a priori* knowledge, in order to obtain a Voronoi-based fuzzy system that solves the particular problem, as done before. But after that, neural network based algorithms can be used to fine tune the performance of the evolved controller based on input output examples. This simple extension can push further the power of the Voronoi-based fuzzy system, allowing better quality controllers to be obtained. Note that considering the temporal extensions to the back propagation algorithm, training can also be performed on the recurrent version of the Voronoi-based fuzzy system.

In all models considered in the thesis, the local approximators were defined as linear functions, however, there is no formal restriction on that. In fact, experiments where carried with the Voronoi approximator in function approximation problems by using quadratic and RBF local approximators, always with the geometric based approach [82]. The basic properties of the model are maintained, meaning that the extension of Voronoi approximation models to fuzzy systems is still possible, with the only difference that now the consequents of the rules are non linear models.

There is also a large area for optimization by considering the computational geometry foundations of the model, since a number of extensions of the Voronoi diagram and Delaunay triangulations have been proposed in the literature. Just to cite a few examples, some extensions attempt to minimize the number of triangles in the approximation, while still retaining a good accuracy in the computation (see for example the Anisotropic Voronoi diagram [91]). In the evaluation of a Voronoi-based fuzzy system, this idea can be implemented as a mutation operator that reduces the rule set by dropping some of the rules, still keeping a good

approximation. Other approaches proposed in the literature remove bad shaped regions, for example, long and very thin areas, by just updating the approximations defined by neighbor Voronoi regions (see for example the extended Delaunay tessellations [67]). In the evolution of our model, a mutation operator can be defined to remove redundant rules from the rule set, enhancing the understandability of the model since regions that potentially produce badly shaped membership functions are removed from the approximation.

As a summary, the Voronoi-based fuzzy system is a powerful and interesting approach, which provides a solid base for further enhancements. With no doubts, large merits go to evolutionary algorithms, which provide a strong platform for developments like the models presented in this thesis.

Bibliography

[1] János Abonyi. *Fuzzy modeling identification for control*. Birkhauser, 2003.

[2] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[3] Guner Alpaydtn, Gunhan Dundar, and Sina Balktr. Evolution-based design of neural fuzzy networks using self-adapting genetic parameters. *IEEE Transactions on Fuzzy Systems*, 10(2):211–221, 2002.

[4] Peter Angeline. Genetic programming's continued evolution. In Peter Angeline and Kenneth Kinnear, editors, *Advances in genetic programming: volume 2*, pages 1–20. MIT Press, 1996.

[5] Peter Angeline. Two self-adaptive crossover operators for genetic programming. In Peter Angeline and Kenneth Kinnear, editors, *Advances in genetic programming: volume 2*, pages 89–109. MIT Press, 1996.

[6] Jim Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In *International Conference of Genetic Algorithms*, pages 86–91. Morgan Kaufmann, June 1989.

[7] I. Babuska and J. L. Melenk. The partition of unity finite element method. Technical Report EN-1185, Institute of Physical Science and Technology, University of Maryland, 1995.

[8] Robert Babuška. Fuzzy modeling: Principles, methods and applications. In C. Bonivento, C. Fantuzzi, and R. Rovatti, editors, *Fuzzy Logic Control: Advances in Methodology*, pages 187–220. World Scientific, Singapore, 1998.

[9] Thomas Back. Introduction to evolutionary algorithms and their instances. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[10] Thomas Back, David Fogel, Darrell Whitley, and Peter Angeline. Mutation. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[11] Thomas Back, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1:1–15, May 1997.

[12] Thomas Back, Gunter Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society, 1993.

[13] Thomas Back and M. Shutz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In *Proceedings of the 4th International Conference on Evolutionary Programming*, pages 33–51. MIT Press, 1995.

[14] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC(13):834–846, 1983.

[15] Jeffrey Bassett and Kenneth De Jong. Evolving behaviors for cooperating agents. In Z. Ras and S. Oshuga, editors, *Proceedings of the ISMIS 2000*, pages 157–165. Springer Verlag, 2000.

[16] Jeffrey Bassett, Mitchell Potter, and Kenneth De Jong. Looking under the EA hood with price's equation. In Kalyanmoy Deb et al., editor, *Proceedings of the Genetic and Evolutionary Computation GECCO*, volume 3103, pages 914–922. Springer-Verlag, Lecture Notes in Computer Science, 2004.

[17] Eric Benoit. *Capteurs symboliques et capteurs flous : un nouveau pas vers l'intelligence*. PhD thesis, Universite Joseph Fourier, Grenoble, 1993.

[18] Eric Benoit, Gilles Mauris, and Laurent Foulloy. Fuzzy sensor aggregation: Application to comfort measurement. In *Proceedings of the IPMU*, pages 721–726, 1994.

[19] Eric Benoit, Gillis Mauris, and Laurent Foulloy. A fuzzy colour sensor. In *Proceedings of the 13th IMEKO*, pages 1015–1020, 1994.

[20] Hamid Berenji and Pratap Khedkar. Learning and tuning fuzzy controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–740, 1992.

[21] Jim Bezdek. Fuzzy models-what are they and why? *IEEE Transactions on Fuzzy Systems*, 1(1):1–6, 1993.

[22] Christopher Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[23] Lashon Booker, David Fogel, Darrell Whitley, and Peter Angeline. Recombination. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[24] Rodney Brooks. Intelligence without reason. AI Lab memo 1293, MIT, April 1991.

[25] Erick Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Springer, 2000.

[26] Purificacion Carinena, Carlos Regueiro, Abraham Otero, Alberto Bugarin, and Senen Barro. Landmark detection in mobile robotics using fuzzy temporal rules. *IEEE Transactions on Fuzzy Systems*, 12(4):423–435, 2004.

[27] Doo-Hyun Choi and Se-Young Oh. A new mutation rule for evolutionary programming motivated from backpropagation learning. *IEEE Transactions on Evolutionary Computation*, 4(2):188–190, July 2000.

[28] Oscar Cordón, Francisco Herrera, and Pedro Villar. Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on Fuzzy Systems*, 9(4):667–674, 2001.

[29] Oscar Cordï¿½, Fernando Gomide, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. Ten years of genetic fuzzy systems: Current framework and new trends. *Fuzzy Sets and Systems*, 141:5–31, 2004.

[30] Nichael Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of the International Conference on Genetic Algorithms and their Applications*, pages 183–187. Carnegie-Mellon University, Pittsburgh, USA, 1985.

[31] M. de Berg, M. van Kreveld, M. OVermars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer Verlag, 1998.

[32] Kenneth de Jong. *Evolutionary Computation: a Unified Approach*. MIT Press, 2006.

[33] Kalyanmoy Deb. Introduction to representations of evolutionary computation models. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[34] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, 2001.

[35] Kalyanmoy Deb and R. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[36] Julie Dickerson and Bart Kosko. Fuzzy function approximation with ellipsoidal rules. *IEEE Transactions on Systems, Man and Cybernetics*, 26(4):542–560, 1996.

[37] Federico Divina and Elena Marchiori. Evolutionary concept learning. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, pages 343–350, 2002.

[38] Dimitris Dracopoulos. *Evolutionary Learning Algorithms for Neural Adaptive Control*. Springer, 1997.

[39] A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35:1–16, 1998.

[40] Agoston Eiben. Evolutionary computing: the most powerful problem solver in the universe? In *Dutch Mathematical Archive*, volume 5/3, pages 126–131. Vrije Universiteit, Amsterdam, 2002.

[41] Agoston Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[42] Agoston Eiben and Marc Schoenauer. Evolutionary computing. *Information Processing Letters*, 82:1–6, 2000.

[43] Agoston Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[44] Larry Eshelman. Genetic algorithms. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[45] Shehu Farinwata, Dimitar Filev, and Reza Langari, editors. *Fuzzy Control, Synthesis and Analysis*. Wiley, 2000.

[46] Gang Feng. An approach to adaptive control of fuzzy dynamic systems. *IEEE Transactions on Fuzzy Systems*, 10(2):268–275, 2002.

[47] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 26(3):396–407, 1996.

[48] David Fogel. Asymptotic convergence properties for genetic algorithms and evolutionary programming: Analysis and experiments. *Cybernetic Systems*, 25:389–407, 1994.

[49] David Fogel. Principles of evolutionary processes. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[50] Alex Freitas. *A survey of evolutionary algorithms for data mining and knowledge discovery*. Springer Verlag, 2003.

[51] Sylvie Galichet and Laurent Foulloy. Fuzzy controllers: Synthesis and equivalences. *IEEE Transactions on Fuzzy Systems*, 3(2):140–148, 1995.

[52] Josselin Garnier, Leila Kallel, and Marc Schoenauer. Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7(2):167–203, 1999.

[53] Jelena Godjevac. Comparative study of fuzzy control, neural network control and neuro-fuzzy control. In D. Ruan, editor, *Fuzzy Set Theory and Advanced Mathematical Applications*, pages 291–322. Kluwer, 1995.

[54] David Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[55] John Grefenstette. The evolution of strategies for multi-agent environments. *Adaptive Behavior*, 1:65–89, 1992.

[56] John Grefenstette. Proportional selection and sampling algorithms. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[57] John Grefenstette. Rank-based selection. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[58] Rudolph Gunter. Evolution strategies. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[59] Nikolaus Hansen, Sibylle Mller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolution Computation*, 11(1), 2003.

[60] Nkolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[61] Simon Haykin. *Neural Networks. A Comprehensive Foundation*. Prentice Hall, New Jersey, USA, 1999.

[62] Jukka Hekanaho. Dogma, a GA based relational learner. In Springer Verlag, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446, pages 205–214. Morgan Kauffman, 1998.

[63] Frank Hoffmann. Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE*, 89(9):1318–1333, 2001.

[64] Mehrdad Hojati and Saeed Gazor. Hybrid adaptive fuzzy identification and control of nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 10(2):211–221, 2002.

[65] Abdollah Homaifar and Ed McCormick. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(2):129–139, 1995.

[66] Martin Hulse, Bruno Lara, Frank Pasemann, and Ulrich Steinmetz. Evolving neural behavior control for autonomous robots. In *Proceedings of the ICANN, Lecture Notes in Computer Science*, volume 2130, pages 957–962. Springer-Verlag, 2001.

[67] Sergio Idelsohn, Eugenio Onate, Nestor Calvo, and Facundo del Pin. The meshless finite element method. *International Journal for Numerical Methods in Engineering*, 58(4), 2003.

[68] Jyh-Shing Jang. Anfis: Adaptive network based inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):665–685, 1993.

[69] Cezary Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.

[70] Keneth De Jong and William Spears. An overview of evolutionary computation. In *Proceedings of European Conference on Machine Learning*, pages 442–259, 1993.

[71] Kenneth De Jong. *An Analysis of the Behavior of a class of genetic adaptive systems*. PhD. thesis, Michigan University, 1975.

[72] Kenneth De Jong and William Spears. On the state of evolutionary computation. In *Proceedings of the ICGA*, pages 618–625. Morgan Kaufmann, 1993.

[73] Kenneth De Jong, William Spears, and Diana Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13(2-3):161–188, 1993.

[74] Chai-Feng Juang and Chin-Teng Lin. A recurrent self-organizing neural fuzzy inference network. *IEEE Transactions on Neural Networks*, 10(4):828–845, 1999.

[75] Chia-Feng Juang. A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 10(2):155–170, 2002.

[76] Chia-Feng Juang, Jiann-Yow Lin, and Chin-Teng Lin. Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Transactions on Systems, Man and Cybernetics*, 30(2):290–302, 2000.

[77] Charles Karr and Edward Gentry. Fuzzy control of ph using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 1(1):46–53, 1993.

[78] Nikola Kasabov and Qun Song. Denfis: Dynamic evolving neural fuzzy inference systems and its application for time series prediction. *IEEE Transactions on Fuzzy Systems*, 10(2):144–154, 2002.

[79] Carlos Kavka, María Liz Crespo, Marcelo Cena, Wu Geng Feng, and Fu Zhong Quian. Fuzzy systems generation through symbiotic evolution. In E. Alpaydin, editor, *Proceedings of the International Symposium on Engineering of Intelligent Systems EIS*, volume 1, pages 37–44. ICS, 1998.

[80] Carlos Kavka, María Liz Crespo, Wu Geng Feng, and Fu Zhong Quian. A fuzzy controllers development tool based on evolutionary techniques. In Peter Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation CEC*, volume 3, pages 2145–2150. IEEE Press, 1999.

[81] Carlos Kavka, Patricia Roggero, and Marc Schoenauer. Evolution of Voronoi based fuzzy recurrent controllers. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO, ACM Press*, pages 1385–1392, Washington, USA, 2005.

[82] Carlos Kavka and Marc Schoenauer. Voronoi diagrams based function identification. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO, Lecture Notes in Computer Science*, pages 1089–1100, Chicago, USA, 2003.

[83] Carlos Kavka and Marc Schoenauer. Evolution of Voronoi-based fuzzy controllers. In *Proceedings of the Conference on Parallel Problem Solving from Nature PPSN, Lecture Notes in Computer Science*, pages 541–550, Birmingham, UK, 2004.

[84] Claire Kennedy and Christophe Giraud-Carrier. An evolutionary approach to concept learning with structured data. In *Proceedings of the Fourth International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 1–6. Springer Verlag, 1999.

[85] Sami Khuri, Thomas Back, and Jorg Heitkotter. The zero/one multiple knapsack problem and genetic algorithms. In *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, pages 188–193. ACM Press, 1994.

[86] Kenneth Kinnear, Robert Smith, and Zbigniew Michalewicz. Derivative methods. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[87] Seong-Gon Kong and Bart Kosko. Adaptive fuzzy systems for backing up a truck-and-trailer. *IEEE Transactions on Neural Networks*, 3(2):211–223, 1992.

[88] Bart Kosko. *Neural Networks and Fuzzy Systems: A dynamical systems approach to machine intelligence*. Prentice Hall, 1992.

[89] Bart Kosko. Fuzzy systems as universal approximators. *IEEE Transactions on Computers*, 43(11):1329–1333, 1994.

[90] John Koza. Genetic programming. In James Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998.

[91] Francois Labelle and Jonathan Schewchuk. Anisotropic Voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proceedings of the Symposium on Computational Geometry, ACM Press*, pages 191–200, USA, 2003.

[92] Pier Luca Lanzi and Rick Riolo. A roadmap to the last decade of learning classifier systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, pages 33–61. Springer Verlag, 2000.

[93] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems, From Foundations to Applications*. Springer-Verlag, London, UK, 2000.

[94] Ching-Hung Lee and Ching-Cheng Teng. Identification and control of dynamic systems using recurrent fuzzy neural networks. *IEEE Transactions on Fuzzy Systems*, 8(4):349–366, 2000.

[95] Chuen Lee. Fuzzy logic in control systems: Fuzzy logic controller - part I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, March/April 1990.

[96] Chuen Lee. Fuzzy logic in control systems: Fuzzy logic controller - part II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):419–435, March/April 1990.

[97] Michael Lee and Hideynki Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, pages 612–617. IEEE, 1993.

[98] D. Leitch. *A New Genetic Algorithm for the Evolution of Fuzzy Systems*. PhD thesis, Department of Engineer Science, University of Oxford, 1995.

[99] Chin-Teng Lin and C.S. George Lee. *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Prentice Hall, 1986.

[100] D. Linkens and H. Nyongesa. Fuzzy modeling: Principles, methods and applications. In *Proceedings of the Control Theory Applications Conference*, volume 143, pages 367–386. IEE, 1996.

[101] Michael Margaliot and Gideon Langholz. *New Approaches to Fuzzy Modeling and Control - Design and Analysis*, chapter One. World Scientific, 2000.

[102] Paris Mastorocostas and John Theocharis. A recurrent fuzzy-neural model for dynamic system identification. *IEEE Transactions on Systems, Man and Cybernetics*, 32(2):176–190, 2002.

[103] P. McQuesten. *Cultural Enhancement of Neuroevolution*. PhD thesis, The University of Texas at Austin, 2002.

[104] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 3rd edition, 1997.

[105] Zbigniew Michalewicz. Introduction to search operators. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[106] Zbigniew Michalewicz, Robert Hinterding, and Maciej Michalewicz. Evolutionary algorithms. In W. Pedrycz, editor, *Fuzzy Evolutionary Computation*, chapter 2. Kluwer Academic, 1997.

[107] Zbigniew Michalewicz, Thomas Logan, and Swarnalatha Swaminathan. Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 84–97. World Scientific Publishing, 1994.

[108] Zbigniew Michalewicz, Girish Nazhiyath, and Machej Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Evolutionary Programming*, pages 305–312. MIT Press, 1996.

[109] Zbigniew Michalewicz, Girish Nazhiyath, and Maciej Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 305–312. MIT Press, 1994.

[110] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[111] O. Michel. Kephera simulator package version 2.0: Freeware mobile robot simulator written at the university of nice sophia-antipolis by olivier michel. Downloadable from the World Wide Web at http://wwwi3s.unice.fr/ om/khep-sim.html.

[112] Sanya Mitaim and Bart Kosko. The shape of fuzzy sets in adaptive function approximation. *IEEE Transactions on Fuzzy Systems*, 9(4):637–656, 2001.

[113] A. Mitchell. *The Finite Element Method in Partial Differential Equations*. John Wiley and Sons, New York, 1977.

[114] S. Mitra and Y. Hayashi. Neuro-fuzzy rule generation: Survey in soft computing framework. *IEEE Transactions on Neural Networks*, 11(5):748–768, 2000.

[115] David Moriarty and Risto Miikulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[116] David Moriarty, Alan Schultz, and John Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligent Research*, 11:241–276, 1999.

[117] George Mouzouris and Jerry Mendel. Dynamic non-singleton fuzzy logic systems for nonlinear modeling. *IEEE Transactions on Fuzzy Systems*, 5(2):199–208, May 1997.

[118] M. Mucientes, R. Iglesias, C. Regueiro, A. Bugarin, P. Carinena, and S. Barro. Fuzzy temporal rules for mobile robot guidance in dynamic environments. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 31(3):391–398, 2001.

[119] Durga Muni, Prasad Pal, and Jyotirmoy Das. A novel approach to design classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196, 2004.

[120] Hung Nguyen, Nadipuram Prasad, Carol Walker, and Ebert Walker. *A First Course in Fuzzy and Neural Control*. Chapman and Hall/Crc, 2003.

[121] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics, The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 2000.

[122] Tandra Pal and Nikhil Pal. Sogarg: A self-organized genetic algorithm-based rule generation scheme for fuzzy controllers. *IEEE Transactions on Evolutionary Computation*, 7(4):397–415, 2003.

[123] William Porto. Evolutionary programming. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[124] Athula Rajapakse, Kazuo Furuta, and Shunsuke Kondo. Evolutionary learning of fuzzy logic controllers and their adaptation through perpetual evolution. *IEEE Transactions on Fuzzy Systems*, 10(3):309–321, 2002.

[125] Ingo Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1972.

[126] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[127] Stefan Schaal and Christopher Atkeson. Contructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[128] M. Schoenauer, F. Jouve, and L. Kallel. Identification of mechanical inclusions. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*. Springer Verlag, 1997.

[129] Marc Schoenauer. Habilitation dissertation. Equipe Evolution Artificielle et Apprentissage, Ecole Polytechnique, France, 1997.

[130] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2^{nd} edition.

[131] Hans-Paul Schwefel. Advantages (and disadvantages) of evolutionary computation over other approaches. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[132] Hans-Paul Schwefel and Rudolf Gunter. Contemporary evolution strategies. In F. Morana et al, editor, *Advances in Artificial Life*, pages 893–907. Springer Verlag, 1995.

[133] Teo Lian Seng, Marzuki Khalid, and Rubiyah Yusof. Tuning of a neuro fuzzy controller by genetic algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 29(2):226–236, 1999.

[134] Magne Setnes and Robert Babuška. Fuzzy modeling for predictive control. In Shehu Farinwata, Dimitar Filev, and Reza Langari, editors, *Fuzzy Control, Synthesis and Analysis*. Wiley, 2000.

[135] Robert Smith, Thomas Back, and William Spears. Heuristics for parameter settings. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[136] William Spears. Recombination parameters. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[137] William Spears, Kenneth De Jong, Thomas Back, David Fogel, and Hugo de Garis. An overview of evolutionary computation. In *ECML '93: Proceedings of the European Conference on Machine Learning*, pages 442–459. Springer-Verlag, 1993.

[138] N. Sukumar, B. Moran, and T. Belytschko. The natural element method in solid mechanics. *International Journal for Numerical Methods in Engineering*, 43(5):839–887, 1998.

[139] Ya Lei Sun and Meng Joo Er. Hybrid fuzzy control of robotic systems. *IEEE Transactions on Fuzzy Systems*, 12(6):755–765, 2004.

[140] Patrick Surry and Nicholas Radcliffe. Formal algorithms + formal representations = search strategies. In Hans-Michel Voight, Werner Ebeling, Ingo Rechenberger, and Hans-Paul Schwefel, editors, *Proceedings of the 3th Parallel Problem Solving from Nature*, volume 1141, pages 366–375. Springer-Verlag, Lecture Notes in Computer Science, 1996.

[141] Patrick Surry and Nicholas Radcliffe. Real representations. In *Foundations of Genetic Algorithms IV*. Morgan Kaufmann, 1996.

[142] Tsuyoshi Takagi and M. Sugeno. Fuzzy identification of systems and its application to modeling. *IEEE Transactions on Systems, Man and Cybernetics*, 15(1):116–132, 1985.

[143] Biing-Tsair Tien and G. Van Straten. The incorporation of qualitative information into T-S fuzzy model. In *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society*, pages 148–153. NAFIP, 1997.

[144] Gancho Vachkov. Adaptive and learning schemes for fuzzy modeling. In Shehu Farinwata, Dimitar Filev, and Reza Langari, editors, *Synthesis and Analysis*, pages 47–72. Wiley, 2000.

[145] Li-Xin Wang and Jerry Mendel. Fuzzy basis functions, universal approximation and orthogonal least squares learning. *IEEE Transactions on Neural Networks*, 3(5):807–814, 1992.

[146] Darrell Whitley. Permutations. In T. Back, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[147] Stewart Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[148] Stewart Wilson and David Goldberg. A critical review of classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255. Morgan Kaufmann, 1989.

[149] D. Withely and J. Kauth. Genitor: A different genetic algorithm. Technical Report CS-88-101, Colorado State University, 1988.

[150] D. Wolpert and W. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, USA, 1996.

[151] Xin Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4(3):203–222, 1993.

[152] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[153] John Yen and Nathan Pfluger. A fuzzy logic based extension to payton and rosenblatt's command fusion method for mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics*, 25(6):971–978, June 1995.

[154] Hao Ying. General Takagi-Sugeno fuzzy systems with linear rule consequent are universal approximators. *IEEE Transactions on Fuzzy Systems*, 6(4):582–587, 1988.

[155] Hao Ying. Constructing nonlinear variable gain controllers via the Takagi-Sugeno fuzzy control. *IEEE Transactions on Fuzzy Systems*, 6(2):226–234, 1998.

[156] Lofti Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[157] Lofti Zadeh. Fuzzy algorithms. *Information and Control*, 12:94–102, 1968.

[158] Lofti Zadeh. Fuzzy logic. *IEEE Computer*, 21(4):83–93, 1988.

[159] Xino-Jun Zeng and Madan Singh. Approximation theory of fuzzy systems-MIMO case. *IEEE Transactions on Fuzzy Systems*, 3(2):219–235, 1995.

[160] Jie Zhang and Julien Morris. Recurrent neuro fuzzy models for nonlinear process modeling. *IEEE Transactions on Neural Networks*, 10(2):313–326, 1999.

[161] O. C. Zienkiewicz. *The Finite Element Method in Engineering Science*. Mc-Graw Hill, New York, 1967.

Index