Tuning Of Fuzzy Systems Using Genetic Algorithms

A thesis submitted to

Prof. Erich Peter Klement

in partial satisfaction of the requirements for the
obtainment of the academic degree of
"Diplom-Ingenieur der Studienrichtung Technische Mathematik"

by

Ulrich Bodenhofer

March 1996

Institut für Mathematik
Johannes Kepler Universität
A-4040 Linz, Austria

E-mail: ulrich@flll.uni-linz.ac.at
WWW: http://www.flll.uni-linz.ac.at/ulrich/my.html

Abstract

Since fuzzy logic has proven to be a very useful tool for representing human knowledge by means of mathematical expressions, the optimization of the involved parameters has been one of the most investigated problems in the theory of fuzzy expert systems. Typically, fuzzy systems have two components – a discrete one, the rules, and a continuous one on the other hand, the so-called fuzzy sets. Very many recent publications concern with the optimization of these two sets of parameters with genetic algorithms (GAs).

Genetic algorithms are optimization methods which are based on the mechanisms of natural evolution, such as selection, mutation, or sexual reproduction. Genetic algorithms were introduced approximately 25 years ago and turned out to be a very promising approach to the solution of many problems in artificial intelligence. During the last years the combination of fuzzy logic and GAs has come into fashion. Nevertheless, or better, for exactly that reason it is necessary to investigate this combination critically and to expose the advantages and weaknesses objectively.

So far, we can distinguish between three classes of combinations. The first one consists of approaches to the tuning of the first component, the fuzzy sets, which represent, in some sense, the semantic information of the rules. This mostly leads to continuous optimization problems with real-valued parameters. The second class comprises methods for the discovery of optimal rulebases. For these cases we typically get optimization problems in discrete, but not necessarily finite spaces. Last, we can collect all the methods, which do not fit in the first two classes, in a third group of methods. In particular, methods, where fuzzy sets and rules are tuned simultaneously, belong to the third type.

This thesis is intended to provide a profound introduction to both fuzzy logic and genetic algorithms and to explore the possibilities to combine the two paradigms.

Acknowledgments

It is not a coincidence that a university study is, in most cases, crowned with the preparation of a thesis. Such a thesis is intended to be an opportunity for the students to prove that they have really understood the things they have learned through the years and to prove that they are really able to apply their knowledge in practice. I hope that I can give these proofs with the present thesis, but I must confess that I would not have come so far without the help of many other persons.

First of all, I want to thank my family not only for financial but, in particular, for mental support and the freedom they left me in my decisions. I never felt that they expected me to do things I actually did not want to do. They always relied on my ambitions. I want to thank them for believing in me.

A further person who deserves my gratefulness is my girlfriend Verena. She helped me to overcome frustrations, which are unavoidable in times of intensive work, and, therefore, kept me from going mad.

Special credits must also be given to many teachers at Braunau grammar school which I attended between '82 and '90. Many of them motivated me for mathematics and for natural science in general. I think they have layed the foundations of my whole career and the pleasure mathematics and related sciences give me now.

On the contrary, it was the environment at the *FLLL* which influenced me most the last years. There were a lot of things I would not have learned or experienced without my colleagues there. I think the discussion and cooperation with a couple of nice, competent colleagues is indispensable for serious scientific work. Especially, I want to thank Peter Bauer and Markus Mittendorfer for showing me what friendship and teamwork can be.

Last, but definitely not least, I want to thank my supervisor Prof. Erich Peter Klement for his trust, support, and encouragement. It is his merit that I could work on an industrial project for a considerable period of time, where I could realize my own ideas there without being interfered too much. Another thing I want to mention is that he made it possible that I could visit some workshops in Austria as well as abroad and that I got in contact with many interesting people from all over the world. However, I think the most important profit for my recent work is that he has shown me how to work scientifically and how to make good presentations.

This thesis is dedicated to all the people who have made it possible.

Ulrich Bodenhofer, March 1996

Contents

List of Figures

Chapter 1

Introduction

> *Growing specialization and diversification have brought a host of monographs and textbooks on increasingly specialized topics. However, the "tree" of knowledge of mathematics and related fields does not grow only by putting forth new branches. It also happens, quite often in fact, that branches which were thought to be completely disparate are suddenly seen to be related.*
>
> Michiel Hazewinkel in the preface of [van Laarhoven and Aarts, 1987]

Many, if not most books and theses on fuzzy logic start referring to the impressive success of this technique in the last decade filling whole pages with lists of successful applications. Since this is sufficiently known, it can be omitted here. I think it is more important to take a look at the challenges of the future than to exult over the success of the presence and the past. Although fuzzy methods have turned out to be very useful tools in many fields, such as control theory, expert systems, cognitive problems, signal- and image processing, robotics, to mention just a few of them, there are many questions which are still to be clarified.

In a rough outline, fuzzy systems are rule-based systems which are capable of dealing with imprecise information. Their advantage is that nearly everything inside the system can be kept interpretable for humans. Thus, prototyping can be done very easy and fast in many cases. Unfortunately, it can take a lot of time to tune all the involved parameters — a problem which certainly becomes worse with increasing complexity of the system. Since the trend turns more and more to the application of fuzzy methods to very complex problems, it is necessary to deal with methods for performing this optimization (semi)automatically. So far, we can roughly distinguish between two categories of methods:

1. **Offline optimization:** Search for parameters such that a given quality measure is optimal; in many cases this quality measure is the discrepancy between the actually obtained output and the desired output for some representative (finite) set of inputs for which the desired or correct output is known.

2. **Online learning:** In the narrow sense, fuzzy systems are nothing else but static input-output functions with the advantage that the parametrization is linguistically interpretable. So, there is definitely no learning capability included in conventional fuzzy systems. On the contrary, there are a lot of problems for which it is not so trivial or even impossible to define a global quality measure, which is required for an offline optimization as described above. Additionally, it can be of interest to design fuzzy systems which adapt to varying circumstances automatically. This entails the demand for adaptive (learning) fuzzy methods.

A class of optimization methods which have come into fashion during the last decades are genetic algorithms (GAs). They can be outlined as "evolutionary methods",as methods which imitate, in some sense, the mechanisms of evolution such as sexual reproduction, mutation, etc.

During the last ten years the number of publications concerning with the application of genetic algorithms to the optimization of fuzzy systems has increased strongly. Papers on offline optimization with genetic algorithms can be found as well as a few publications about online learning systems which use GAs for the adaptation operations. The present thesis gives a survey of the various approaches.

How this Thesis is Organized

Chapter 2 provides a comprehensive introduction to the concepts and methods behind fuzzy systems. The methods, which are really used in the applications discussed later, are especially emphasized. Nevertheless, the theoretical background is also exposed as much as necessary and meaningful.

Similarly, chapter 3 provides an introduction to the theory of genetic algorithms. At first, different concepts and variants are discussed followed by a brief convergence analysis. The chapter is closed with a view to hybrid methods and other extensions.

Chapters 4–6 represent the real main part of this thesis. Based on the considerations given in chapter 2 and 3, they give a structured overview and comparative analysis of actual possibilities to combine fuzzy logic and genetic algorithms. While chapter 4 deals with the optimization of the

shape of fuzzy sets with GAs, chapter 5 gives an introduction to the field of methods for the acquisation of rulebases. Chapter 6 closes the thesis with a view to further ideas combining fuzzy logic and genetic algorithms.

Appendix A is intended to be a reference work. Some mathematical symbols and notations, which occur differently in the world of mathematics, are defined there in order to avoid misunderstandings. Moreover, a brief introduction to probability theory is given there to provide the theoretical framework for the convergence analysis of genetic algorithms in chapter 3.

Technical Background of this Work

This text was typesetted using LaTeX 2_ε with a customized document class based on the LaTeX 2_ε standard document class `report`. N. Schwarz's `dc` fonts were used as standard fonts. The work was additionally supported by $\mathcal{AMS}$ fonts and symbols and the packages `array`, `epsfig`, `exscale`, `ipa`, `makeidx`, and `rotating`. The bibliography database was created and maintained with BibTeX. The index was created with the immens help of the `makeindex` program. Pictures and graphics were created with *Mathematica*, `xfig`, or other tools and included as encapsulated PostScript files with the `epsfig` package. Finally, the text was printed on a laser writer with a resolution of 600 dpi.

Example programs were written in `C` on *SiliconGraphics* workstations using the Irix-C-compiler with X11-Motif and graphics libraries.

Chapter 2

The Fuzzy Rule-Based Approach to Artificial Intelligence

> **fuzzy** \'fəz-ē\ *adj* **fuzz·i·er; -est 1:** covered with or resembling fuzz **2:** not clear: INDISTINCT [perhaps from Low German *fussig* "loose, spongy"] — **fuzz·i·ly** \'fəz-ə-lē\ *adv* — **fuzz·i·ness** \'fə-z-ē-nəs\ *n*
>
> Webster's New Encyclopedic Dictionary

Seriously, the goal of artificial intelligence is not to build androids. The main aim of artificial intelligence has always been to construct programs and/or devices which perform human-like operations, decisions, etc. in order to speed up production, to eliminate humans as factors of insecurity, or just to give humans support in their decisions and actions.

The first approaches to artificial intelligence were (crisp) expert systems and models imitating the functionality of the human brain — so-called artificial neural networks (ANNs). While conventional expert systems are rule-based systems, which manipulate symbolic expressions based on traditional binary logic, neural nets are systems with, in most cases, real-valued input and output, which can learn from sample data, but without the possibility to survey the actions inside. From that point of view, fuzzy logic can be seen as an alternative to the two paradigms mentioned above. Why this is the case should become clear within the next pages.

In a rough outline, fuzzy systems are rule-based systems which are able to process vague, imprecise data. In this sense, fuzzy systems can be regarded

as generalized rule-based expert systems. Obviously, this generalization requires a mathematical formulation of impreciseness and inference methods adapted to this model.

2.1 Fuzzy Sets

2.1.1 The Basics

As anticipated above, we must define a mathematical framework for impreciseness which should, in some sense, be an extension of binary logic. Let us start with an example where mathematical preciceness is inappropriate and unnatural. Consider for instance how to define the set of old people. If we want to express "old" with a conventional set, it would be necessary to fix a certain limit above people are old and below people are not old, respectively. If we fix 70 as limit why should it then be justified to say that a 71 year old person is old, but, on the other hand, a 69 year old person is not old? Obviously, this is not the understanding of the term "old" humans have. Fuzzy logic overcomes this problem by allowing intermediate degrees of membership. This concept was introduced by L. A. Zadeh in 1965 (see [Zadeh, 1965]). The following definition can be regarded as the basis of fuzzy logic:

Definition 2.1 Let X be an arbitrary set, the so-called universe of discourse. Then a mapping $\mu : X \longrightarrow [0,1]$ is called a fuzzy subset of X. We will often abbreviate this with "fuzzy set". The set of fuzzy subsets (the fuzzy powerset) of X is denoted with $\mathcal{F}(X) := X^{[0,1]}$. A fuzzy set μ is called normalized if and only if $\exists x \in X : \mu(x) = 1$.

The values $\mu(x)$ represent the degrees of membership to which the points x belong to the fuzzy set μ. Of course, a membership value of 0 means that an x does definitely not belong to the fuzzy set μ and a value of 1 means that x certainly belongs to μ.

In classical set theory the characteristic function of a set M is defined as

$$\begin{array}{rccl} \chi_M : & X & \longrightarrow & \{0,1\} \\ & x & \longmapsto & \begin{cases} 0 & \text{if } x \notin M \\ 1 & \text{if } x \in M \end{cases} \end{array} \tag{2.1}$$

Obviously, there is a bijective relationship between subsets $M \subseteq X$ and functions $f : X \to \{0,1\}$.

$$\begin{array}{rccc} i : & P(X) & \longrightarrow & X^{\{0,1\}} \\ & M \subseteq X & \longmapsto & \chi_M \end{array}$$

From this point of view, subsets and characteristic functions can be identified with each other.

Analogously, we speak of a fuzzy set A and its corresponding membership function μ_A, although, in this case, exactly the same object is meant.

Furthermore, it is clear from the definitions above that conventional so-called crisp sets are special fuzzy sets whose membership values are either 0 or 1, but nothing in between.

In order to calculate with fuzzy sets we must generalize the basic set operations such as union and intersection. The intersection and the union of two crisp sets is given by

$$\begin{aligned} A \cap B &:= \{x \in X | x \in A \wedge x \in B\}, \\ A \cup B &:= \{x \in X | x \in A \vee x \in B\}, \end{aligned}$$

or formulated with characteristic functions

$$\begin{aligned} \chi_{A \cap B}(x) &:= \chi_A(x) \wedge \chi_B(x), \\ \chi_{A \cup B}(x) &:= \chi_A(x) \vee \chi_B(x). \end{aligned} \tag{2.2}$$

From these formulas it is obvious that union and intersection are nothing else but the application of $\wedge$ and $\vee$ to the membership values $\chi_A(x)$ and $\chi_B(x)$ for each x. So, for defining a fuzzy intersection and a fuzzy union, it is near at hand to define generalized logical operators and to apply them analogously to (2.2).

Definition 2.2 A t-norm (triangular norm) is a binary operation on the unit interval $T : [0,1]^2 \longrightarrow [0,1]$ which satisfies the following axioms:

$\forall x, y \in [0,1]:$	$T(x,y) = T(y,x)$	(Commutativity)
$\forall x, y, z \in [0,1]:$	$T(x, T(y,z)) = T(T(x,y), z)$	(Associativity)
$\forall x, y, y' \in [0,1]:$	$y \leq y' \Longrightarrow T(x,y) \leq T(x,y')$	(Monotonicity)
$\forall x, y \in [0,1]:$	$T(x,1) = x \wedge T(1,y) = y$	(Boundary condition)

Similarly, a binary operation $S : [0,1]^2 \longrightarrow [0,1]$ is called a t-conorm (triangular conorm) if the conditions

$\forall x, y \in [0,1]:$	$S(x,y) = S(y,x)$	(Commutativity)
$\forall x, y, z \in [0,1]:$	$S(x, S(y,z)) = S(S(x,y), z)$	(Associativity)
$\forall x, y, y' \in [0,1]:$	$y \leq y' \Longrightarrow S(x,y) \leq S(x,y')$	(Monotonicity)
$\forall x, y \in [0,1]:$	$S(x,0) = x \wedge S(0,y) = y$	(Boundary condition)

are fulfilled. It is easy to see that t-norms and t-conorms interpolate the classical AND and OR.

Remark 2.3 Due to the associativity of t-norms and t-conorms, it is justified to consider t-norms and t-conorms as n-ary operations.

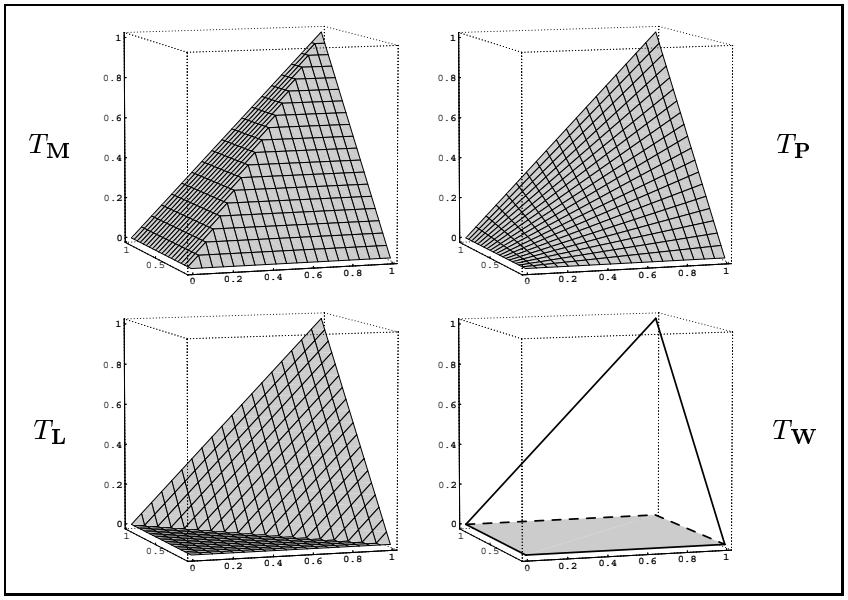


Figure 2.1: Commonly used t-norms

Example 2.4 Figure 2.1 shows graphs of four important representatives of t-norms. The first one is

$$T_{\mathbf{M}}(x, y) := \min(x, y), \tag{2.3}$$

which was introduced by L. A. Zadeh himself in his epoch-making paper [Zadeh, 1965]. The second one

$$T_{\mathbf{L}}(x, y) := \max(x + y - 1, 0) \tag{2.4}$$

goes to back to J. Łukasiewicz, a Polish pioneer of multivalued logic. Very popular in many applications due to its smoothness is the product:

$$T_{\mathbf{P}}(x, y) := x \cdot y \tag{2.5}$$

Rather unusual for applications but of theoretical interest is the so-called drastic t-norm:

$$T_{\mathbf{W}}(x, y) := \begin{cases} y & \text{if } x = 1 \\ x & \text{if } y = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

It can be proven easily that $T_{\mathbf{M}}$ is the biggest and that $T_{\mathbf{W}}$ is the smallest t-norm.

In fact, there are infinitely many different t-norms and lots of classes or parameterized families of t-norms, more information can be found for

example in [Butnariu and Klement, 1993], [Kruse *et al.*, 1994] or [Geyer-Schulz, 1995].

Definition 2.5 Analogously to (2.2), we can now define the fuzzy intersection with respect to a t-norm T and the fuzzy union with respect to a t-conorm S:

$$\begin{aligned} \mu_{A\cap_T B}(x) &:= T(\mu_A(x), \mu_B(x)) \\ \mu_{A\cup_S B}(x) &:= S(\mu_A(x), \mu_B(x)) \end{aligned} \tag{2.7}$$

Again we can consider the union and the intersection as n-ary operations. In the following we will often use expressions, such as

$$\bigcup_{i=1}^{n}{}_{S} A_i,$$

where the union is set to A_1 if $n = 1$; analogously for the intersection.

Figure 2.2 shows how the shape of the intersection is influenced by the t-norm.

Another question, which must also be clarified in this framework, is how to define the complement of a fuzzy set. It is not so hard to see that this is related to the definition of a generalized logical negation.

Definition 2.6 The complement $\mathsf{C}A$ of a fuzzy set A is defined as

$$\mu_{\mathsf{C}A}(x) := 1 - \mu_A(x). \tag{2.8}$$

In the world of fuzzy logic $\bar{A}$ is commonly used for the fuzzy complement. The notation $\mathsf{C}A$ was chosen just for consistency and to avoid confusions with the topological closure.

Remark 2.7 Of course, other generalized negations are reasonable. In [Geyer-Schulz, 1995] a fuzzy negation is defined as an arbitrary non-increasing function $n : [0, 1] \to [0, 1]$ which fulfills some additional boundary conditions. The reason that only $1 - x$ was defined here is that it is almost the only one which is really used in practice.

An important questions is, of course, how many properties of classical binary logic are preserved in the fuzzy case. One equality, which is very useful for simplifying expressions, is the so-called De-Morgan law. Its classical formulation is $\neg(a \wedge b) = \neg a \vee \neg b$. Obviously, the fuzzy equivalent is

$$1 - T(x, y) = S(1 - x, 1 - y). \tag{2.9}$$

It can be seen easily that this formula itself is a criterion for checking if a t-norm/t-conorm pair satisfies the De-Morgan law. Moreover, from this formula a suitable t-conorm S for a given t-norm T can be derived with $S(x, y) := 1 - T(1 - x, 1 - y)$ and vice versa.

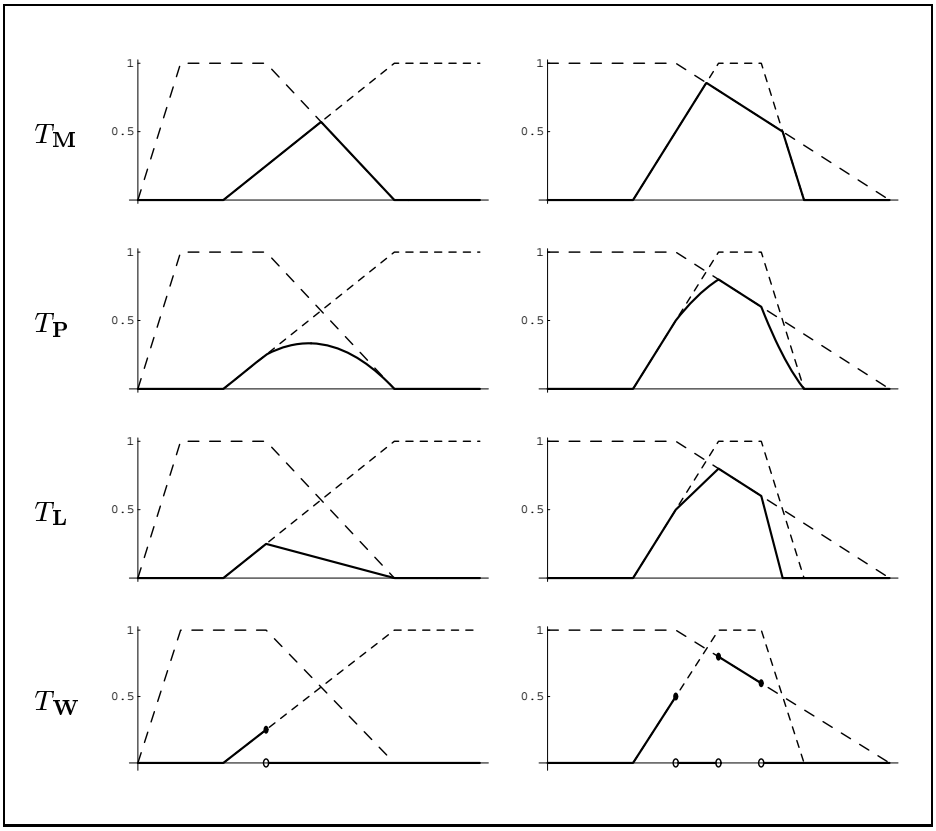


Figure 2.2: Fuzzy intersections with respect to different t-norms

Example 2.8 The following table shows suitable t-conorms for the four examples shown in 2.4:

Minimum $T_{\mathbf{M}}$	Maximum $S_{\mathbf{M}}$
$T_{\mathbf{M}}(x,y) = \min(x,y)$	$S_{\mathbf{M}}(x,y) := \max(x,y)$
Product $T_{\mathbf{P}}$	Probabilistic Sum $S_{\mathbf{P}}$
$T_{\mathbf{P}}(x,y) = x \cdot y$	$S_{\mathbf{P}}(x,y) := x + y - x \cdot y$
Łukasiewicz t-norm $T_{\mathbf{L}}$	Bounded Sum $S_{\mathbf{L}}$
$T_{\mathbf{L}}(x,y) = \max(x+y-1,0)$	$S_{\mathbf{L}}(x,y) := \min(x+y,1)$
Drastic t-norm $T_{\mathbf{W}}$	Drastic Sum $S_{\mathbf{W}}$
$T_{\mathbf{W}}(x,y) = \begin{cases} y & \text{if } x = 1 \\ x & \text{if } y = 1 \\ 0 & \text{otherwise} \end{cases}$	$S_{\mathbf{W}}(x,y) := \begin{cases} y & \text{if } x = 0 \\ x & \text{if } y = 0 \\ 1 & \text{otherwise} \end{cases}$

The proofs that the De-Morgan law really holds for these pairs are left to the reader.

2.1.2 More about Operations on Fuzzy Sets

Definition 2.9 Under the assumption that A and B are fuzzy subsets of universes of discourse X and Y, respectively, the fuzzy Cartesian product of A and B in the product space $X \times Y$ with respect to t-norm T is defined as

$$\mu_{A\times_T B}(x, y) := T(\mu_A(x), \mu_B(y)). \tag{2.10}$$

Let us denote the set of all fuzzy Cartesian products of fuzzy subsets of X and Y with $\mathcal{F}(X) \times_T \mathcal{F}(Y) := \{A \times_T B | A \in \mathcal{F}(x) \wedge B \in \mathcal{F}(Y)\}$.

Note that $\mathcal{F}(X)\times_T\mathcal{F}(Y) \subseteq \mathcal{F}(X\times Y)$ but for any pair X, Y with $|X| > 1$ and $|Y| > 1$ a trivial example shows that $\mathcal{F}(X) \times_T \mathcal{F}(Y) \neq \mathcal{F}(X \times Y)$.

Definition 2.10 For a fixed $\alpha \in [0, 1]$ the crisp set

$$A^{\geq\alpha} := \{x \in X | \mu_A(x) \geq \alpha\} \tag{2.11}$$

is called α-cut of fuzzy set A. Analogously, we define the set

$$A^{>\alpha} := \{x \in X | \mu_A(x) > \alpha\} \tag{2.12}$$

which is called strict α-cut of fuzzy set A.

In some application it can be necessary to transform one or more parameters. If these parameters are given in a fuzzy form, the transformation must be applied to fuzzy sets. This leads us to the definition of the image of a fuzzy set with respect to a given crisp function ϕ. This idea is called extension principle and goes back to L. A. Zadeh too.

Definition 2.11 Let X be an arbitrary universe of discourse and let $\phi : X \to Y$ be a function. Then the extension of ϕ to $\mathcal{F}(X)$ is defined in the following way:

$$\begin{array}{rccc} \hat{\phi}: & \mathcal{F}(X) & \longrightarrow & \mathcal{F}(Y) \\ & A & \longmapsto & \hat{\phi}(A) \end{array}$$

where

$$\begin{array}{rccc} \mu_{\hat{\phi}(A)}: & Y & \longrightarrow & [0, 1] \\ & y & \longmapsto & \sup\{\mu_A(x) | y = \phi(x)\}. \end{array} \tag{2.13}$$

The following theorem gives a representation of the strict α-cuts of the extension which are actually the images of the strict α-cuts of the original fuzzy subset.

Theorem 2.12

$$\forall \phi \in X^Y \quad \forall A \in \mathcal{F}(X) \quad \forall \alpha \in [0,1]: \quad \hat{\phi}(A)^{>\alpha} = \phi(A^{>\alpha}) \tag{2.14}$$

Proof: Let ϕ, A, and $\alpha \in [0,1)$ (trivial for $\alpha = 1$) be fixed. Then

$$\begin{aligned}\hat{\phi}(A)^{>\alpha} &= \{y \in Y \,|\, \sup\{\mu_A(x) | y = \phi(x)\} > \alpha\} \\ &= \{y \in Y \,|\, \exists x \in X: \quad y = \phi(x) \wedge \mu_A(x) > \alpha\} \\ &= \{y \in Y \,|\, \exists x \in A^{>\alpha}: \quad y = \phi(x)\} = \phi(A^{>\alpha}).\end{aligned}$$

■

The extension can also be generalized to Cartesian products.

Definition 2.13 If we have universes of discourse $(X_i)_{i \in \overline{1,n}}, Y$, corresponding fuzzy subsets $(A_i)_{i \in \overline{1,n}}$ with $A_i \in \mathcal{F}(X_i)$, and a function $\psi : \bigotimes_{i=1}^{n} X_i \to Y$, then the extension of ψ to $\mathcal{F}(X_1) \times_T \cdots \times_T \mathcal{F}(X_n)$ with respect to t-norm T is defined as

$$\begin{array}{rccl}\hat{\psi}: & \mathcal{F}(X_1) \times_T \cdots \times_T \mathcal{F}(X_n) & \to & \mathcal{F}(Y) \\ & A_1 \times_T \cdots \times_T A_n & \mapsto & \hat{\psi}(A_1 \times_T \cdots \times_T A_n)\end{array}$$

with

$$\begin{aligned}\mu_{\hat{\psi}(A_1 \times_T \cdots \times_T A_n)}(y) := \sup\{T(\mu_{A_1}(x_1), \ldots, \mu_{A_n}(x_n)) | \\ y = \psi(x_1, \ldots, x_n)\}.\end{aligned} \tag{2.15}$$

Again a representation with strict α-cuts can be proven:

Theorem 2.14 *Under the assumptions of definition 2.13, the extension, which was defined there, matches exactly the result of applying the extension principle defined in 2.11 in the product space. Then 2.12 yields for any* $\alpha \in [0,1]$

$$\hat{\psi}(A_1 \times_T \cdots \times_T A_n)^{>\alpha} = \psi([A_1 \times_T \cdots \times_T A_n]^{>\alpha}). \tag{2.16}$$

If $T = T_{\mathbf{M}}$*, we can conclude further that*

$$\hat{\psi}(A_1 \times_T \cdots \times_T A_n)^{>\alpha} = \psi(A_1^{>\alpha} \times \cdots \times A_n^{>\alpha}). \tag{2.17}$$

Proof: If we apply the extension principle 2.11 in the product space, we get

$$\begin{aligned}\mu_{\hat{\psi}(A_1 \times_T \cdots \times_T A_n)}(y) &= \sup\{\mu_{A_1 \times_T \cdots \times_T A_n}(x_1, \ldots, x_n) | y = \psi(x_1, \ldots, x_n)\} \\ &= \sup\{T(\mu_{A_1}(x_1), \ldots, \mu_{A_n}(x_n)) | y = \psi(x_1, \ldots, x_n)\}\end{aligned}$$

for an arbitrary $y \in Y$ and we have shown the consistency. Then (2.16) follows directly from theorem 2.12.

Now suppose that $T = T_{\mathbf{M}}$, then

$$\begin{aligned}[A_1 \times_T \cdots \times_T A_n]^{>\alpha} &= \{(x_1, \ldots, x_n) | \min(\mu_{A_1}(x_1), \ldots, \mu_{A_n}(x_n)) > \alpha\} \\ &= \{(x_1, \ldots, x_n) | \forall i \in \overline{1,n} : \ \mu_{A_i}(x_i) > \alpha\} \\ &= A_1^{>\alpha} \times \cdots \times A_n^{>\alpha}\end{aligned}$$

and we get, together with (2.16),

$$\hat{\psi}(A_1 \times_T \cdots \times_T A_n)^{>\alpha} = \psi(A_1^{>\alpha} \times \cdots \times A_n^{>\alpha}).$$

■

2.2 Fuzzy Reasoning

As already anticipated previously, our goal is to formalize a rule-based paradigm which is capable of dealing with imprecise information. Since we have already defined fuzzy sets as an appropriate model for impreciseness, it is now time to discuss concepts for dealing with fuzzy information in a rule-based way.

2.2.1 Rulebases

Crisp rule-based expert systems normally consist of rules of the following shape:

$$\mathsf{IF}\ A\ \mathsf{THEN}\ \ B, \tag{2.18}$$

where B is an arbitrary statement which is executed if and only if the logical expression A is true; A is called the antecedent, premise, or condition, B is called consequence. The rule (2.18) can be regarded as an implementation of the so-called classical modus ponens.

$$(A,\ A \Longrightarrow B) \quad \models \quad B \tag{2.19}$$

As we are mainly discussing expert systems, decision systems, etc., it is sufficient to restrict to rulebases with a finite number of rules m of the following kind:

$$\begin{array}{llll} \mathsf{IF} & L_1(x_1\ \mathsf{is}\ A_{11}, \ldots, x_n\ \mathsf{is}\ A_{1n}) & \mathsf{THEN} & y := b_1 \\ \vdots & \vdots & \vdots & \vdots \\ \mathsf{IF} & L_m(x_1\ \mathsf{is}\ A_{m1}, \ldots, x_n\ \mathsf{is}\ A_{mn}) & \mathsf{THEN} & y := b_m \end{array} \tag{2.20}$$

where $(x_i)_{i \in \overline{1,n}}$ are the input variables taken from the input spaces $(X_i)_{i \in \overline{1,n}}$, $(A_{ji})_{j \in \overline{1,m}}$ are subsets of the corresponding input space X_i, y is the output

variable taken from the output space Y, $(b_j)_{j\in\overline{1,m}}$ are values in Y, and $(L_j)_{j\in\overline{1,m}}$ are n-ary logical expressions. Note that X_i can be arbitrary sets such as real intervals, sets of natural numbers, and, of course, which is very usual in decision systems, sets of linguistic labels.

Before we turn to the fuzzy case we have to analyze the semantics of of the rulebase above in more detail. As apparent from (2.20), the rulebase is, in some sense, a function from a subset M of $X_1 \times \cdots \times X_n$ to a subset N of Y. The statement "x_i is A_{ji}" is a more readable notation for the truth value of $x_i \in A_{ji}$ which is, of course, $\chi_{A_{ji}}(x_i)$. With the setting $R_j := \{(x_1, \ldots, x_n) | L_j(\chi_{A_{j1}}(x_1), \ldots, \chi_{A_{jn}}(x_n)) = 1\}$ we can define the semantics of the rulebase, the input-output function Φ according to the rulebase, respectively:

$$\begin{array}{rlcll} \Phi: & M \subseteq \bigotimes_{i=1}^{n} X_i & \longrightarrow & N \subseteq Y & \\ & (x_1, \ldots, x_n) & \longmapsto & b_j & \text{if } (x_1, \ldots, x_n) \in R_j \end{array} \tag{2.21}$$

Obviously, Φ is well-defined if and only if

$$\bigcup_{j=1}^{m} (M \cap R_j) = M \quad \text{and} \quad \forall k \neq j: \; R_k \cap R_j \cap M \neq \emptyset \Rightarrow b_k = b_j.$$

The rules, the premises of which are true for a given input, are called firing rules.

The fuzzification of the inference discussed above poses the following problems for us:

1. Since we want to model transitions, we must allow more than one firing rule. Consequently, we have to find a method to aggregate the outputs of the different firing rules.

2. In the crisp case the transition from the premise of a rule to the output, the so-called inference, is nothing else but the classical modus ponens. Since we also want to deal with truth values which are not necessarily 0 or 1, it is necessary to have concepts which generalize the classical modus ponens. This generalization is often called approximate reasoning.

3. In some applications it can be required to process a fuzzy set as input instead of a single value.

2.2.2 Mamdani Inference

First of all, we should define an exact notion for a "fuzzy variable" (see also [Zadeh, 1987a] and [Zadeh, 1987b]).

Definition 2.15 A linguistic variable is a quintuple of the form

$$(A, T(A), U, G, M), \tag{2.22}$$

where A, $T(A)$, U, G, and M are defined as follows:

1. A is the name of the linguistic variable
2. $T(A)$ is the ground set of verbal values of A
3. U is the universe of discourse of variable A
4. G is the definition of the grammar which produces the names of the values of A
5. M is a semantic rule which maps every verbal value $X \in T(A)$ to its meaning (fuzzy subset of U) $M(X)$

This definition provides a generality which is an overkill in many cases. In the following, we will often suffice with a simplified variant:

Definition 2.16 A fuzzy variable is a tuple $V = (X, \mathcal{A})$, where X is a crisp set, the universe of discourse of variable V, and $\mathcal{A}$ is a finite set of fuzzy subsets of X. We assume implicitely that there are unique verbal labels for the variable and the fuzzy sets.

The first step towards the fuzzification of rulebases is, of course, to replace the input variables in (2.20) by fuzzy variables in the sense of 2.16.

If we replace the consequent values b_j by one-elementary sets $B_j := \{b_j\}$, we get an equivalent formulation of (2.20) which can be fuzzified in a straightforward way. The procedure of computing the output of the rulebase $\Phi(x_1, \ldots, x_n)$ can then be implemented as follows:

$B := \emptyset$;

FOR $j := 1$ **TO** m **DO**
  IF $L_j(\chi_{A_{j1}}(x_1), \ldots, \chi_{A_{jn}}(x_n))$ **THEN**
    $B := (B \cup B_j)$;

As apparent from this algorithm, the output set is the union of all those output sets B_j for which $L_j(\chi_{A_{j1}}(x_1), \ldots, \chi_{A_{jn}}(x_n)) = 1$. With other words

$$B = \bigcup_{j=1}^{m} C_j(x_1, \ldots, x_n), \tag{2.23}$$

with the definition

$$\chi_{C_j(x_1,\ldots,x_n)}(y) := \begin{cases} L_j(\chi_{A_{j1}}(x_1),\ldots,\chi_{A_{jn}}(x_n)) & \text{if } y = b_j \\ 0 & \text{otherwise,} \end{cases} \tag{2.24}$$

i.e.

$$\chi_{C_j(x_1,\ldots,x_n)}(y) = \chi_{B_j}(y) \wedge L_j(\chi_{A_{j1}}(x_1),\ldots,\chi_{A_{jn}}(x_n)). \tag{2.25}$$

From this point of view, the degree of membership, to which b_j belongs to the final output set B, is exactly the truth value of the expression $L_j(x_1,\ldots,x_n)$. Obviously, in this formulation we can allow an arbitrary number of firing rules, even if they have different consequences. This procedure can now be fuzzified easily.

We consider the following fuzzy rulebase:

$$\begin{array}{llll} \mathsf{IF} & \tilde{L}_1(x_1 \text{ is } A_{11},\ldots,x_n \text{ is } A_{1n}) & \mathsf{THEN} & y \text{ is } B_1 \\ \vdots & \vdots & \vdots & \vdots \\ \mathsf{IF} & \tilde{L}_m(x_1 \text{ is } A_{m1},\ldots,x_n \text{ is } A_{mn}) & \mathsf{THEN} & y \text{ is } B_m, \end{array} \tag{2.26}$$

where $\tilde{L}_j$ are (fuzzy) logical expressions, $(x_i = (X_i, \mathcal{A}_i))_{i\in\overline{1,n}}$ and $y = (Y, \mathcal{B})$ are fuzzy variables, A_{ij} and B_j are fuzzy subsets of X_i and Y with $A_{ij} \in \mathcal{A}_i$ and $B_j \in \mathcal{B}_j$, respectively.

In order to formulate a semantical interpretation of (2.26) we first need a t-norm T_1 and the suitable t-conorm S_1 for evaluating the logical expressions $\tilde{L}_j$, a t-norm T_2, and a t-conorm S_2 for computing the output set. Then we can write down the corresponding input-output function explicitely:

$$\begin{array}{rlcl} \Psi: & M \subseteq \bigotimes_{i=1}^{n} X_i & \longrightarrow & \mathcal{F}(Y) \\ & (x_1,\ldots,x_n) & \longmapsto & \Psi(x_1,\ldots,x_n) := \left(\bigcup_{j=1}^{m}\right)_{S_2} C_j(x_1,\ldots,x_n), \end{array} \tag{2.27}$$

with

$$\begin{array}{rlcl} \mu_{C_j(x_1,\ldots,x_n)}: & Y & \longrightarrow & [0,1] \\ & y & \longmapsto & T_2(\mu_{B_j}(y), \tilde{L}_j(\mu_{A_{j1}}(x_1),\ldots,\mu_{A_{jn}}(x_n))), \end{array} \tag{2.28}$$

what is obviously a fuzzification of (2.25). This approach was invented by E. H. Mamdani (see [Mamdani and Assilian, 1975] and [Mamdani, 1977]), a pioneer of fuzzy control. Thus, this method is called Mamdani inference.

2.2.3 Generalizations

In this paragraph more general concepts of fuzzy inference are presented. They are based, in particular, on the notion of so-called fuzzy relations.

In classical logic a relation on $X \times Y$ is a logical predicate of the form $x \sim y :=$ *logical expression*. Obviously, this predicate can be regarded as the characteristic function of a subset of $X \times Y$. Hence, the following generalization is near at hand:

Definition 2.17 Assume that X and Y are arbitrary universes of discourse, then a fuzzy subset R is called a fuzzy relation on $X \times Y$. The value $\mu_R(x, y)$ can be regarded as the degree to which x is R-related to y.

Of course, many notions and definitions of the traditional relational calculus can be taken over in a more or less straightforward way. We mention the most important ones just for the sake of completeness.

Definition 2.18 Let X, Y, and Z be crisp sets, T a given t-norm, and R and Q fuzzy relations on $X \times Y$ and $Y \times Z$, respectively. Then the composition $Q \circ R \in \mathcal{F}(X \times Z)$ is defined as

$$\begin{array}{rccl} \mu_{Q \circ R}: & X \times Z & \longrightarrow & [0,1] \\ & (x,z) & \longmapsto & \sup\{T(\mu_R(x,y), \mu_Q(y,z)) | y \in Y\}. \end{array} \tag{2.29}$$

If R is a relation on $X \times X$ for an arbitrary X, we can define the following properties:

1. R is called reflexive if and only if $\mu_R(x, x) = 1$ for every $x \in X$
2. R is called symmetric if and only if $\mu_R(x, y) = \mu_R(y, x)$ for every pair $(x, y) \in X^2$
3. R is called T-transitive if and only if $T(\mu_R(x, y), \mu_R(y, z)) \leq \mu_R(x, z)$ for every triple x, y and z

A fuzzy relation which fulfills all these three conditions is called a fuzzy equality (relation).

The next thing we have to define in our theoretical investigation of fuzzy inference is the "image" of a fuzzy set with respect to a fuzzy relation R.

Definition 2.19 For given sets X and Y, a fuzzy subset $A \in \mathcal{F}(X)$, a given t-norm T, and a fuzzy relation R on $X \times Y$ the image $R \circ A \in \mathcal{F}(Y)$ of A with respect to relation R is defined as

$$\mu_{R \circ A}(y) := \sup\{T(\mu_A(x), \mu_R(x, y)) | x \in X\} \quad \forall y \in Y. \tag{2.30}$$

As they are intended to be, rulebases represent nothing else but input-output relations. So, it is not so far-fetched to interpret fuzzy rulebases as fuzzy relations. On the other hand, it can be of interest to fix relationships between input and output and to try to find an appropriate fuzzy relation, inference method, respectively, which models exactly these relationships.

Without loss of generality, we can restrict to rulebases with only one input variable. The reason is that a rulebase with n inputs $(X_i, \mathcal{A}_i)_{i=\overline{1,n}}$ can be rewritten easily as a rulebase with one input $(X_1 \times \cdots \times X_n, \tilde{\mathcal{A}})$ where the fuzzy subsets in $\tilde{\mathcal{A}}$ are intersections and unions of Cartesian products of fuzzy subsets from $\tilde{\mathcal{A}}_i$ with respect to t-norm T_1 and t-conorm S_1.

If we want to represent the fuzzy rule

$$\mathsf{IF} \quad x \text{ is } A \quad \mathsf{THEN} \quad y \text{ is } B$$

with a fuzzy relation as close as possible, we have to find a fuzzy relation R on $X \times Y$ such that the relational equation

$$R \circ A = B \tag{2.31}$$

is fulfilled. For m rules we get a system of relational equations

$$R \circ A_i = B_i, \quad i \in \overline{1,m} \tag{2.32}$$

which has to be solved.

Theorem 2.20 *For a relational equation of the form (2.31), where A is a normalized fuzzy subset of X, $R = A \times_T B$ is the exact solution (see also [Bauer* et al.*, 1995] for comparison).*

Proof: We have to show that $(A \times_T B) \circ A = B$. Let $y \in Y$ be arbitrary but fixed, then

$$\begin{aligned}
\mu_{(A\times_T B)\circ A}(y) &= \sup\{T(\mu_A(x), \mu_{A\times_T B}(x,y) | x \in X\} \\
&= \sup\{T(\mu_A(x), T(\mu_A(x), \mu_B(y))) | x \in X\} \\
&= \sup\{\underbrace{T(T(\mu_A(x), \mu_A(x)), \mu_B(y))}_{=:(*)} | x \in X\}.
\end{aligned}$$

t-norms are monotonic in each component, thus $(*)$ takes its supremum in that point $\bar{x}$ where μ_A also takes its supremum. A was assumed as normalized, hence we get

$$\mu_{(A\times_T B)\circ A}(y) = T(T(1,1), \mu_B(y)) = \mu_B(y),$$

which completes the proof. ■

For systems of more than one relational equation the existence of a solution is not guaranteed in such a simple way. The following relation is an approximate solution of the system (2.32):

$$R := \bigcup_{i=1}^{m}{}_{S_2}(A_i \times_{T_2} B_i) \tag{2.33}$$

The question is now how to compute the output of a rulebase which is given in the relational form.

Definition 2.21 Let R be a fuzzy relation on $X \times Y$. Then the fuzzy constraint $R|_{x_0} \in \mathcal{F}(Y)$ of R with respect to $x_0 \in X$ is defined as

$$\mu_{R|_{x_0}}(y) := \mu_R(x_0, y) \quad \forall y \in Y. \tag{2.34}$$

If R is a fuzzy relation representing a rulebase, then the fuzzy output of the rulebase for a given $x_0 \in X$ is defined as the constraint $R|_{x_0}$ of R with respect to x_0.

The next theorem closes the loop to the previous section.

Theorem 2.22 *If we compute the output of a rulebase with one input variable $(X, \mathcal{A})$ and the output variable $(Y, \mathcal{B})$ and rules of the form*

$$\mathsf{IF} \quad x \ \mathsf{is}\ A_i \quad \mathsf{THEN} \quad y \ \mathsf{is}\ B_i$$

using the Mamdani inference, the input-output function is exactly

$$\begin{array}{rccc} \phi: & X & \longrightarrow & \mathcal{F}(Y) \\ & x & \longmapsto & R|_x, \end{array} \tag{2.35}$$

where the fuzzy relation R is defined as in (2.33).

Proof: With the definitions of (2.27) and (2.28) we can write down the Mamdani inference for our case as

$$\begin{array}{rccl} \Psi: & X & \longrightarrow & \mathcal{F}(Y) \\ & x & \longmapsto & \Psi(x) := C_1(x) \cup_{S_2} \cdots \cup_{S_2} C_m(x) \end{array}$$

with

$$\begin{array}{rccl} \mu_{C_j(x)}: & Y & \longrightarrow & [0,1] \\ & y & \longmapsto & T_2(\mu_{B_j}(y), \mu_{A_j}(x)) \end{array}$$

We have to show that for every $x_0 \in X$ the equation

$$\underbrace{\bigcup_{j=1}^{m}{}_{S_2} C_j(x)}_{=:D} = \left(\bigcup_{j=1}^{m}{}_{S_2}(A_j \times_{T_2} B_j) \right)\Bigg|_{x_0}$$

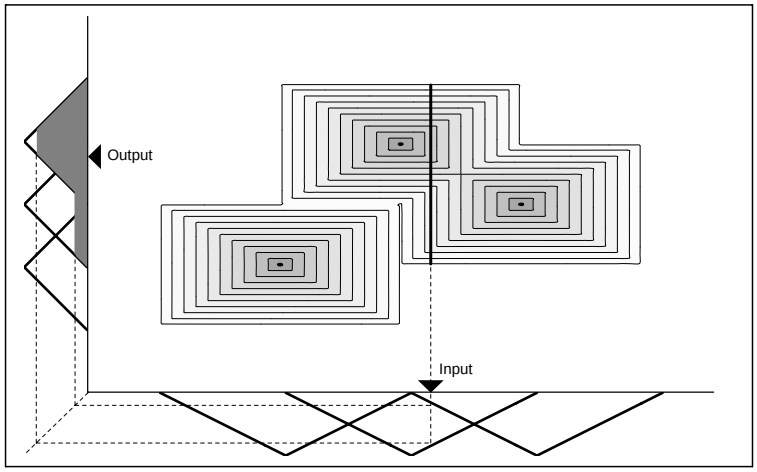


Figure 2.3: Geometrical interpetration of the Mamdani inference (taken from [Bauer *et al.*, 1995])

holds. Let $y \in Y$ be arbitrary but fixed. Then the equation

$$\begin{aligned}\mu_{R|_{x_0}}(y) &= \mu_R(x_0, y) \\ &= S_2(\underbrace{T_2(\mu_{A_1}(x_0), \mu_{B_1}(y))}_{=\mu_{C_1(x_0)}(y)}, \ldots, \underbrace{T_2(\mu_{A_m}(x_0), \mu_{B_m}(y))}_{=\mu_{C_m(x_0)}(y)}) \\ &= \mu_D(y),\end{aligned}$$

holds, which completes the proof. ■

Figure 2.3 shows a geometrical interpretation of this fact.

As apparent from

$$\begin{aligned}\mu_{R\circ\{x_0\}}(y) &= \sup\{T(\chi_{\{x_0\}}(x), \mu_R(x, y)) | x \in X\} \\ &= T(\chi_{\{x_0\}}(x_0), \mu_R(x_0, y)) = \mu_R(x_0, y) = \mu_{R|_{x_0}}(y),\end{aligned}$$

the equation

$$R|_{x_0} \equiv R \circ \{x_0\} \tag{2.36}$$

holds. If we replace the one-elementary set $\{x_0\}$ by an arbitrary fuzzy set, we get a method for processing fuzzy input.

Definition 2.23 For a rulebase with one input variable $(X, \mathcal{A})$ and the output variable $(Y, \mathcal{B})$, where the rulebase is given in the form of the fuzzy relation $R \in \mathcal{F}(X \times Y)$, the image of a fuzzy set $A \in \mathcal{F}(X)$ is defined as

$$R \circ A.$$

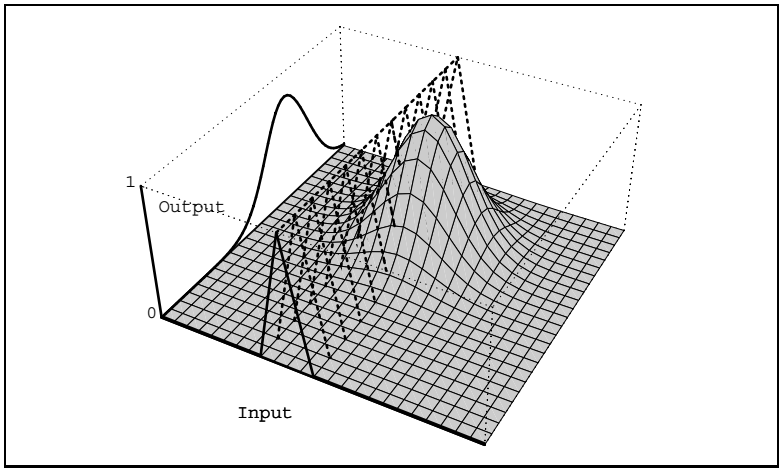


Figure 2.4: Example for the application of the compositional rule of inference (taken from [Bauer *et al.*, 1995])

For $R \circ A$ we have the explicit form

$$\mu_{R \circ A}(y) = \sup\{T_2(\mu_A(x), \mu_R(x, y)) | x \in X\}.$$

This formula can also be interpreted geometrically (see also [Bauer *et al.*, 1995]) as the composition of the following operations

1. cylindric extension of A to $A \times Y$

$$\mu_{A \times_{T_2} Y}(x, y) = \mu_A(x)$$

2. intersection of $A \times Y$ with the relation R

$$\begin{aligned} \mu_{(A \times_{T_2} Y) \cap_{T_2} R}(x, y) &= T_2(\mu_{A \times_{T_2} Y}(x, y), \mu_R(x, y)) \\ &= T_2(\mu_A(x), \mu_R(x, y)) \end{aligned}$$

3. projection of this intersection onto Y

$$\begin{aligned} \mu_B(y) &:= \sup\{\mu_{(A \times_{T_2} Y) \cap_{T_2} R}(x, y) | x \in X\} \\ &= \sup\{T_2(\mu_A(x), \mu_R(x, y)) | x \in X\} = \mu_{R \circ A}(y) \end{aligned}$$

This algorithm is called the compositional rule of inference; Figure 2.4 shows a schematical visualization of the compositional rule of inference for a simple example.

Concluding remarks

The previous pages dealing with theoretical aspects of fuzzy inference were intended to provide an introduction to the fields, which are required later, with a reasonable amount of generality. Of course, there are many more aspects and different treatments of approximate reasoning. For further approaches based on relations and equality relations see [Bauer, 1994], [Kruse *et al.*, 1994], and [Bauer *et al.*, 1995]. For details on possibilistic approaches we refer to [Kruse *et al.*, 1994], [Geyer-Schulz, 1995], and some papers by R. R. Yager, [Yager, 1980] and [Yager, 1983].

2.3 Processing the Output of a Rulebase

As defined above, the output of rulebases consists of fuzzy sets. This is undesired in many applications. Mainly, there are two types of applications where postprocessing of the output is required. On the one hand, in certain types of expert systems, such as diagnosis systems, the output should be a verbal expression, a linguistic label. We call the process of searching for verbal labels for arbitrary fuzzy sets linguistic approximation. On the other hand, especially in control applications, it may be necessary to have crisp values as output. The methods, which compute a representative value of a fuzzy set, are called defuzzification methods.

2.3.1 Linguistic Approximation

First of all, take a closer look at the definition of a linguistic variable

$$(A, T(A), U, G, M).$$

In many applications $T(A)$ is a finite set of linguistic labels such as "small", "large", or "hot". $M : T(A) \to \mathcal{F}(U)$ is the function which maps each linguistic label to its meaning — a fuzzy subset of U. So, M may be an injective mapping, but, since $T(A)$ is, at most, a countable set, but $\mathcal{F}(U)$ is always an uncountable one, it cannot be surjective. The aim of linguistic approximation is to find a generalized "inverse" function $\tilde{M}$ which gives linguistic interpretations of arbitrary fuzzy subsets of U (see also [Geyer-Schulz, 1995]).

The Best Fit Method

This method can be applied if $T(A)$ is a finite set. The main idea of the best fit method is to use a certain measure of discrepancy between two fuzzy sets. For a given fuzzy subset B of the domain U the linguistic label of an

arbitrary fuzzy subset of U is determined as the linguistic label of the fuzzy set in $T(A)$ with the least distance to B:

$$\tilde{M}(B) := M^{-1}(C) \quad \text{with} \quad \tilde{d}(B,C) = \inf\{\tilde{d}(B,D) | D \in T(A)\}, \qquad (2.37)$$

where $\tilde{d}(.,.)$ is a pseudometric on $\mathcal{F}(U)$.

Other Approaches

If the grammar G consists of more than only atomic expressions, so-called primary terms, the exhaustive best fit method may be unapplicable because G can be defined recursively with the consequence that $T(A)$ can be an infinite set. In such cases, the following grammatical elements are commonly used (compare with [Geyer-Schulz, 1995]):

Adjectives: atomic expressions such as "small" or "heavy"

Adverbs: modifiers such as "above", "very", or "not"; they can be realized by logical operations but they can also be given as transformations of the adjectives (see 2.1.2)

Connectives: binary (logical) connectives, such as "and", "or", "but", etc. They are realized as unions, intersections, etc.

These elements can, for instance, be combined using the following grammar

<verbal expression⟩ := <adjective> |
"(" <adverb> <verbal expression> ")" |
"(" <verbal expression> <connective> <verbal expression> ")";

For this case there are two approaches well-known so far. One is the method of successive approximation which is based on refining the approximation by applying more and more modifiers recursively. Further details can be found in [Geyer-Schulz, 1995]. The second method is based on the egmentation of a fuzzy set into several parts. Then the parts are interpreted independently. Finally, these parts are joined together to a whole approximation by applying the connectives. This is called the method of piecewise decomposition. It can be found in [Geyer-Schulz, 1995], [Eshrag and Mamdani, 1979], or [Novák, 1989].

2.3.2 Defuzzification

In lots of applications the output of a rulebase should not be a verbal answer or a fuzzy set but a certain control such as a force or a current. So, it can

be necessary to have one representative value of the output set. Here we restrict to universes of discourse which are finite intervals. This is not a serious restriction since forces, temperatures, or currents cannot take arbitrary values in practice. Let O be a fuzzy subset of the domain $Y = [a, b]$ in the following (see also [Kruse *et al.*, 1994]).

Mean of Maximum Method (MOM)

The mean of maximum method concentrates on the area where the degree of membership is maximal. If the set

$$\mathrm{Ceil}(O) := \{y \in Y | \mu_O(y) = \sup_{x \in Y} \mu_O(x)\} \tag{2.38}$$

is non-empty and measurable, the MOM defuzzification of O is defined as

$$\xi_{\mathrm{MOM}}(O) := \frac{\int\limits_{\mathrm{Ceil}(O)} y \; dy}{\int\limits_{\mathrm{Ceil}(O)} 1 \; dy} \tag{2.39}$$

Center of Gravity Method (COM)

A more global strategy, which also takes areas with lower degree of membership into account, is the center of gravity (COG) method. It mostly yields better analytical properties of the output function in terms of smoothness. It is defined as the first coordinate of the center of gravity of the area under the membership function. Under the assumption that μ_O is an integrable function, the COG defuzzification of O is defined as

$$\xi_{\mathrm{COG}}(O) := \frac{\int\limits_a^b y \cdot \mu_O(y) \; dy}{\int\limits_a^b \mu_O(y) \; dy}. \tag{2.40}$$

Center of Area Method (COA)

The center of area method (COA) takes that point which separates the area under the membership function into equally sized parts. If μ_O is an

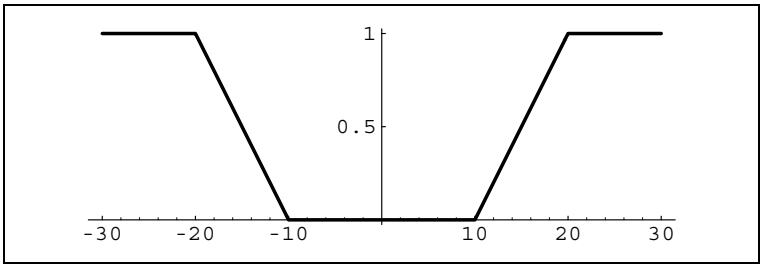


Figure 2.5: A fuzzy set where the MOM, COG, and COA method may yield undesired results

integrable function, the COA defuzzification of O is defined as mean value of $\bar{x}_1$ and $\bar{x}_2$, where

$$\begin{aligned} \bar{x}_1 &:= \inf\{x \in Y = [a,b] | \int_a^x \mu_O(y)\ dy = \int_x^b \mu_O(y)\ dy\} \\ \bar{x}_2 &:= \sup\{x \in Y = [a,b] | \int_a^x \mu_O(y)\ dy = \int_x^b \mu_O(y)\ dy\}. \end{aligned} \tag{2.41}$$

Max Criterion Methods

Figure 2.5 shows a reasonable output for the steering angle of an object which has to be moved around without colliding with obstacles. This set can be interpreted as "evade the obstacle either by turning to the left or to the right". Obviously, each of the three methods discussed above would yield 0 as defuzzified value what means, in fact, crashing directly into the obstacle.

The max criterion methods are a class of methods which have in common that they choose an element of the set $\mathrm{Ceil}(O)$ by applying some random selection. The disadvantage of this approach is unpredictable, in some cases instable behavior.

An alternative could be the following technique which is applicable whenever $\mathrm{Ceil}(O)$ is a union of finitely many disjoint intervals $([a_i, b_i])_{i \in \overline{1,N}}$:

$$\xi_{\mathrm{MMC}}(O) := \begin{cases} \frac{a_j + b_j}{2} & \text{if } |C(O)| = 1 \\ \frac{a_k + b_k}{2} & \text{with } [a_k, b_k] \text{ chosen} \\ & \text{randomly from } C(O) \text{ if } |C(O)| > 1 \end{cases} \tag{2.42}$$

where $C(O) := \{[a_l, b_l] | l \in \overline{1,N},\ b_l - a_l = \max_{i \in \overline{1,N}}(b_i - a_i)\}$ is the set of intervals with maximal length. We call this modified max criterion method (MMC).

2.4 Fuzzy Systems

Since we have discussed all the essential ingredients of fuzzy systems, a term, which we have used more or less intuitively until now, we can now join these together in order to define exactly what a fuzzy system is. Roughly, fuzzy systems are input-output systems which incorporate the methods we have already dealt with. In literature there is nearly no exact definition "fuzzy system" given. Nevertheless, it would be nice to have an exact, compact definition and a mathematical representation of what a fuzzy system is. Of course, "fuzzy system" is a term which can be interpreted in various ways. So, the following definitions are only one possibility. They were chosen such that all the elements, which we need in the following, are provided.

Definition 2.24 An atomic fuzzy system with fuzzy output is a quintuple of the form

$$(X, V, \phi, O, R_{T_1;S_1;T_2;S_2}), \tag{2.43}$$

where

1. X is a family of n input spaces $(X_1, \ldots, X_n)$,
2. V is a family of $\bar{n}$ linguistic variables $(A_i, T_i(A_i), U_i, G_i, M_i)_{i \in \overline{1,\bar{n}}}$,
3. ϕ is a preprocessing function from $X_1 \times \cdots \times X_n$ to $U_1 \times \cdots \times U_{\bar{n}}$
4. $O = (B, T(B), U_B, G_B, M_B)$ is the output variable,
5. $R_{T_1;S_1;T_2;S_2}$ is a rulebase whose semantics can be interpreted as a fuzzy relation on $U_1 \times \cdots \times U_{\bar{n}} \times U_B$. Inside, the logical expressions are evaluated using the operations T_1 and S_1, the inference is performed applying T_2 and S_2 as discussed in 2.2.

Note that such a system is capable of dealing with fuzzy input as well, since we have discussed how to extend the function ϕ to fuzzy subsets of $X_1 \times \cdots \times X_n$.

The output of such a system is then defined as

$$\begin{array}{rccl} \Phi: & \bigotimes_{i=1}^{n} & \longrightarrow & \mathcal{F}(U_B) \\ & (x_1, \ldots, x_n) & \longmapsto & R_{T_1;S_1;T_2;S_2} \circ \{\phi(x_1, \ldots, x_n)\} \end{array} \tag{2.44}$$

for crisp input and as

$$\begin{array}{rccl} \Phi: & \mathcal{F}(\bigotimes_{i=1}^{n}) & \longrightarrow & \mathcal{F}(U_B) \\ & A & \longmapsto & R_{T_1;S_1;T_2;S_2} \circ \hat{\phi}(A) \end{array} \tag{2.45}$$

for fuzzy input.

An atomic fuzzy system with crisp output is defined as a tuple

$$(X, V, \phi, O, R_{T_1;S_1;T_2;S_2}, \xi) \tag{2.46}$$

where the first five entries are defined as above and ξ is a defuzzification method. Then the output is defined as

$$\begin{array}{rccl} \Phi: & \bigotimes_{i=1}^{n} & \longrightarrow & U_B \\ & (x_1, \ldots, x_n) & \longmapsto & \xi\left(R_{T_1;S_1;T_2;S_2} \circ \{\phi(x_1, \ldots, x_n)\}\right) \end{array} \tag{2.47}$$

for crisp input and as

$$\begin{array}{rccl} \Phi: & \mathcal{F}(\bigotimes_{i=1}^{n}) & \longrightarrow & U_B \\ & A & \longmapsto & \xi\left(R_{T_1;S_1;T_2;S_2} \circ \hat{\phi}(A)\right) \end{array} \tag{2.48}$$

for fuzzy input.

Remark 2.25 The concept of the preprocessing function is not commonly used in the world of fuzzy logic. Here it is intended to allow expressions such as

$$\mathsf{IF} \quad x_1 + x_2 \text{ is “low”} \quad \mathsf{THEN} \quad \ldots$$

which can be very useful in many applications.

Now we can turn to more complicated systems. As it is useful in many applications to bundle the information, we consider hierarchical fuzzy systems.

Definition 2.26 A composite (or hierarchical) fuzzy system is a triple of the form

$$(X, F, C), \tag{2.49}$$

where

1. $X = (X_1, \ldots, X_n)$ is a family of n input spaces,
2. $F = (F_1, \ldots, F_N)$ is a set of N pairwise different atomic fuzzy systems, which can have either crisp or fuzzy output,
3. $C = (c_{ij})_{i \in \overline{1,N}, j \in \overline{1,n+N}}$ is a $N \times (n+N)$ connectivity matrix with natural entries, which fulfills the following conditions:
   - Assume that the atomic fuzzy system F_i has n_i inputs, then the i-th line of C must have exactly n_i non-zero entries and the set of non-zero entries must be exactly $\overline{1, n_i}$.

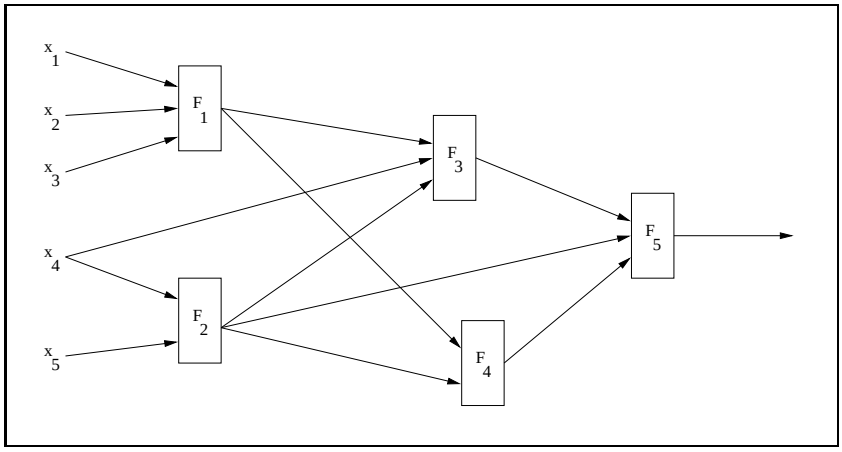


Figure 2.6: A composite fuzzy system

- Two atomic fuzzy systems F_j and F_i are called primarly connected if and only if $c_{i,n+j} \neq 0$. F_j and F_i are called connected if there is a primarly connected sequence $(F_{i_1}, \ldots, F_{i_k})$ with $F_{i_1} = F_j$ and $F_{i_k} = F_i$. There must not be any F_i which is connected to itself.
- If $c_{ij} = a$ for $1 \leq i \leq n$ then the input space with number a of the atomic fuzzy system F_i must be equal to X_j. If $c_{ij} = a$ for $n+1 \leq i \leq n+N$ then the input space with number a of atomic fuzzy system F_i must be the same as the universe of discourse of the output variable of atomic fuzzy system $F_j - n$.

$c_{ij} = K$ means that the K-th input variable of the atomic fuzzy system F_i is either the j-th input variable of the system, if $1 \leq j \leq n$, or, otherwise, the output variable of F_{j-n}. Logically, two atomic fuzzy systems F_j and F_i are primarly connected if the output of F_j is an input of F_i. The output variables of these atomic fuzzy systems, which are not connected to any others, are called output variables of the fuzzy system.

It is intuitively clear that a composite fuzzy system is a directed, cyclefree graph of atomic fuzzy systems. In the following we will call a composite fuzzy system just fuzzy system for abbreviation.

Example 2.27 The connectivity matrix C of the fuzzy system shown in figure 2.6 would be

	x_1	x_2	x_3	x_4	x_5	F_1	F_2	F_3	F_4	F_5
F_1	1	2	3	0	0	0	0	0	0	0
F_2	0	0	0	1	2	0	0	0	0	0
F_3	0	0	0	2	0	1	3	0	0	0
F_4	0	0	0	0	0	1	2	0	0	0
F_5	0	0	0	0	0	0	2	1	3	0

The following lemma guarantees that the output of a composite fuzzy system as defined above is well-defined even if it is not a tree.

Lemma 2.28 *Under the assumptions of definition 2.26, the relation* $F_i \preccurlyeq F_j := (F_i = F_j) \vee (F_i$ *is connected to* $F_j)$ *is a partial order on* R*. The minimal elements with respect to* $\preccurlyeq$ *are those atomic fuzzy systems* F_k *where all the entries in the* k*-th line of* C *are between* 1 *and* n*. Then the output of the fuzzy system can be computed in a well-defined way.*

Proof: Obviously, $\preccurlyeq$ is a reflexive and transitive relation. Assume that there are F_i and F_j which are inequal but $F_i \preccurlyeq F_j$ and $F_j \preccurlyeq F_i$. Then we have two nontrivial primarly connected sequences $F_i \prec \cdots \prec F_j$ and $F_j \prec \cdots \prec F_i$. If we merge these two sequences, we get a nontrivial primarly connected sequence $F_i \prec \cdots \prec F_j \prec \cdots \prec F_i$, which is a contradiction. Thus, $\preccurlyeq$ is also antisymmetric.

From the definition above we see that, if there is no atomic fuzzy system F_j which is connected to a certain F_i, all the entries must be between 1 and n.

Since we are dealing with a partial order on a finite set, there exists at least one minimal element (lemma of Zorn). So, we can compute all the outputs of minimal atomic fuzzy systems. This process can be repeated inductively by removing the minimal elements from the list of systems, whose output still has to be computed, and computing the output of the minimal elements of the remaining list until the list is empty. ■

Algorithm 2.29 Based on the proof of 2.28 we can write down an algorithm which computes the output of a fuzzy system of the form (2.49).

WHILE *there are unmarked atomic fuzzy systems* **DO**
BEGIN
select an unmarked F_i *whose input is already available*;
compute the output of F_i;
mark F_i
END

where "input already available" means that the input of F_i consists of input variables of the system or of outputs of already marked atomic fuzzy systems.

Lemma 2.28 guarantees that this procedure terminates and yields the correct output.

2.5 Fuzzy Control

A very important subclass of fuzzy systems are so-called fuzzy controllers. They are simple fuzzy systems of a special kind which are used in typical control applications. So far, we can distinguish between two important kinds of fuzzy controllers.

2.5.1 Mamdani Controllers

Definition 2.30 An atomic fuzzy system with n crisp inputs, without preprocessing function, and with crisp output

$$(X, V, \mathrm{id}, O, R_{T_1;S_1;T_2;S_2}, \xi_{\mathrm{COG}}) \tag{2.50}$$

is called Mamdani controller if

- for each $V_i = (A_i, T_i(A_i), U_i, G_i, M_i)$ we have $U_i = X_i$ (of course, because we have no preprocessing function),
- the domain U_B of the output variable $O = (B, T(B), U_B, G_B, M_B)$ is a measurable subset of $\mathbb{R}$, an interval in the simpliest case,
- every $M_B(C)$, where $C \in T(B)$, is integrable.

The semantical interpretation of a Mamdani controller as defined above is then

$$\begin{array}{rccl} \Phi: & \bigotimes_{i=1}^{n} & \longrightarrow & U_B \\ & (x_1, \ldots, x_n) & \longmapsto & \frac{\int_{U_B} y \cdot \mu_{R_{T_1;S_1;T_2;S_2}}(x_1,\ldots,x_n,y)\ dy}{\int_{U_B} \mu_{R_{T_1;S_1;T_2;S_2}}(x_1,\ldots,x_n,y)\ dy}. \end{array} \tag{2.51}$$

Unfortunately, the output surface Φ of a Mamdani controller can have very bad properties in terms of smoothness and interpolation behavior.

2.5.2 Sugeno Controllers

The second class of a fuzzy controllers, which is named after M. Sugeno (see [Sugeno, 1985]), is much simplier. It cannot be seen in the context of our considerations on fuzzy systems at first glance. The advantages of this approach are its simplicity and the good smoothness and interpolation properties.

Definition 2.31 A rulebase with m rules of the form

$$\mathsf{IF} \quad x \text{ is } A_j \quad \mathsf{THEN} \quad y \text{ is } f_j(x), \tag{2.52}$$

with the semantical interpretation

$$\begin{array}{rccc} \Psi: & X & \longrightarrow & Y \\ & x & \longmapsto & \dfrac{\sum\limits_{j=1}^{m} \mu_{A_j}(x) \cdot f_j(x)}{\sum\limits_{j=1}^{m} \mu_{A_j}}, \end{array} \tag{2.53}$$

where $f_j : X \to Y$ are arbitrary functions and A_j are fuzzy subsets of the input space X, is called a Sugeno controller.

In many applications f_j are constants. Often they are affine linear functions, linear manifolds. Controllers of this kind are sometimes called Takagi-Sugeno controllers (see [Takagi and Sugeno, 1985]) after T. Takagi and M. Sugeno.

The weighted sum may look like a discrete version of the COG method. Indeed:

Theorem 2.32 *A Sugeno controller with pairwise different constants b_j on the right hand side can be represented as Mamdani controller with operations T_{M} and S_{M}.*

Proof: With the definitions $B_j := [b_j - \varepsilon, b_j + \varepsilon]$, where ε is chosen such that the sets B_j are pairwise disjoint, we can write the input-output function of the Mamdani controller with rulebase

$$\mathsf{IF} \quad x \text{ is } A_j \quad \mathsf{THEN} \quad y \text{ is } B_j$$

as

$$\Phi(x) = \frac{\int\limits_{\mathbb{R}} y \cdot \mu_R(x, y)\ dy}{\int\limits_{\mathbb{R}} \mu_R(x, y)\ dy},$$

where

$$\begin{aligned}\mu_R(x,y) &= S_{\mathbf{M}}(T_{\mathbf{M}}(\mu_{A_1}(x),\mu_{B_1}(y)),\ldots,T_{\mathbf{M}}(\mu_{A_m}(x),\mu_{B_m}(y))) = \\ &= \begin{cases} \mu_{A_j}(x) & \text{if } y \in B_j \\ 0 & \text{otherwise,} \end{cases}\end{aligned}$$

because the sets B_j are disjoint. Hence, we get

$$\begin{aligned}\Phi(x) &= \frac{\sum_{j=1}^{m} \int_{b_j-\varepsilon}^{b_j+\varepsilon} y\cdot\mu_{A_j}(x)\ dy}{\sum_{j=1}^{m} \int_{b_j-\varepsilon}^{b_j+\varepsilon} \mu_{A_j}(x)\ dy} = \frac{\sum_{j=1}^{m} \mu_{A_j}(x) \int_{b_j-\varepsilon}^{b_j+\varepsilon} y\ dy}{\sum_{j=1}^{m} 2\varepsilon\cdot\mu_{A_j}(x)} \\ &= \frac{\sum_{j=1}^{m} \mu_{A_j}(x)\cdot 2\varepsilon\cdot b_j}{2\varepsilon \sum_{j=1}^{m} \mu_{A_j}(x)} = \frac{\sum_{j=1}^{m} \mu_{A_j}(x)\cdot b_j}{\sum_{j=1}^{m} \mu_{A_j}(x)}\end{aligned}$$

and the proof is completed. ■

There are a lot of publications concerning with the properties of Sugeno controllers. L. X. Wang has proven ([Wang, 1992]) that the set of Sugeno controllers with $X = [a_1, b_1] \times \cdots \times [a_n, b_n]$ as input space is dense in the set of continous functions $f : X \to \mathbb{R}$ with respect to the sup norm. B. Moser (see [Moser, 1996]) has shown that the set of Sugeno controllers with $X = [a_1, b_1] \times \cdots \times [a_n, b_n]$ as input space with a bounded number of rules is nowhere dense in the set of continous functions $f : X \to \mathbb{R}$, again with respect to the sup norm. Further works also concerning with these properties are, just for example, [Kosko, 1992], [Buckley, 1993], [Bauer *et al.*, 1993], and [Bauer *et al.*, 1995].

Chapter 3

Genetic Algorithms

> *Although the belief that an organ so perfect as the eye could have been formed by natural selection, is enough to stagger any one; yet in the case of any organ, if we know of a long series of gradations in complexity, each good for its possessor, then, under changing conditions of life, there is no logical impossibility in the acquirement of any conceivable degree of perfection through natural selection.*

Charles Robert Darwin about difficulties of his theory ([Darwin, 1991])

Applying mathematics to a problem of the real world mostly means, at first, modeling the problem mathematically, maybe with hard restrictions, idealizations, or simplifications, then solving the mathematical problem, and finally, drawing conclusions about the real problem based on the mathematical solutions. During the last decades the, in some sense, opposite way has come into fashion — imitating intelligent procedures, which occur in the real world, with mathematical algorithms. Three examples of such algorithms, which can be regarded as based on procedures of nature, are artificial neural networks, genetic algorithms, and a probabilistic optimization method called simulated annealing.

If we disregard religious aspects for a moment, the world as we see it today, with its variety of different creatures, its individuals highly adapted to their environment, with its ecological balance (under the ideal assumptions that there is still one), is the product of a three billion years experiment we call evolution, a process based on sexual and asexual reproduction, selection, mutation, and so on. If we look inside, these procedures are certain operations on the genetic material of the individuals. The complexity and

adaptability of today's creatures has been achieved by refining and combining the genetic material over a long period of time. Genetic algorithms are probabilistic optimization methods which try to imitate this process. This concept was first introduced in 1967 by J. D. Bagley in his PhD thesis "The Behavior of Adaptive Systems Which Employ Genetic and Correlative Algorithms" ([Bagley, 1967]). The theory and applicability was also strongly influenced by J. H. Holland, a pioneer of genetic algorithms. The first textbook on GAs, which has become a standard reference work, was written by D. E. Goldberg ([Goldberg, 1989]). Another useful standard work is the collection [Davis, 1991].

3.1 Concepts and Notions

3.1.1 The Basics

As anticipated above, genetic algorithms are optimization methods based on the mechanisms of natural reproduction which are operations on the genetic material of living beings. This genetic information is contained in the cell nuclei of the sex cells of these creatures. It consists of a certain number of chromosomes which carry the genetic information, the genes. So, for obvious reasons, genetic algorithms operate on structures which are organized similar to chromosomes.

Let us consider the following optimization problem:

$$\begin{array}{l}\text{Find an } x_0 \in X \text{ such that } f \text{ is maximal in } x_0, \text{ where } f : X \to \mathbb{R} \\ \text{is an arbitrary real-valued function, i.e. } f(x_0) = \sup\limits_{x \in X} f(x).\end{array} \tag{3.1}$$

We call X the search space of the uncoded problem; f is called objective function of the optimization problem. Let S be a set of strings and G a grammar which describes the syntax of the strings contained in S. Then the function

$$\begin{array}{rccc} c: & X & \longrightarrow & S \\ & x & \longmapsto & c(x) \end{array} \tag{3.2}$$

and the function

$$\begin{array}{rccc} \tilde{c}: & S & \longrightarrow & X \\ & s & \longmapsto & \tilde{c}(s) \end{array} \tag{3.3}$$

are called coding- and decoding function, respectively, if and only if $\tilde{c}$ is injective and $(c \circ \tilde{c}) \equiv \mathrm{id}_S$; S is called search space of the encoded optimization problem:

$$\begin{array}{l}\text{Find an } s_0 \in S \text{ such that } \tilde{f} := f \circ \tilde{c} \text{ is maximal in } s_0, \\ \text{i.e. } \tilde{f}(s_0) = \sup\limits_{s \in S} \tilde{f}(s).\end{array} \tag{3.4}$$

A genetic algorithm is now a probabilistic optimization method which tries to solve the optimization task (3.4) by applying genetic operations.

The following table lists analogies between natural evolution and the genetic algorithm paradigm:

Natural Evolution	Genetic Algorithm
genotype	coded string
phenotype	uncoded point
chromosome	string
gene	string position
allele	value at a certain position
fitness	objective function value

Of course, a GA can only be a simplified model of the process of evolution. The most important difference is that "fitness" in the real world cannot be expressed by a single value. Seriously, fitness in the real world is a vector with components, such as "intelligence", "strength", or "fertility".

We have seen above that genetic algorithms try to solve a coded (transformed) optimization problem instead of the real problem itself. The second significant difference between conventional methods and genetic algorithms is that GAs do not operate on single points but on whole sets of trial points — a generation (population) of strings. The most general formulation of a GA is the following outline of an algorithm:

Algorithm 3.1

$t := 0$;
Compute $\mathcal{B}_0$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
  $\mathcal{B}_{t+1} := \delta(\mathcal{B}_t)$;
  $t := t + 1$
END

δ is the so-called probabilistic transition operator which computes the next generation $\mathcal{B}_{t+1}$ from the previous one $\mathcal{B}_t$ taking the fitness of its individuals into account.

Normally, δ is a composition of various probabilistic operators which we will discuss in the following.

Compared with traditional continuous optimization methods, such as Newton- or gradient descent methods we can state the following significant differences:

1. GAs manipulate coded versions of the problem parameters instead of the parameters themselves.

2. While almost all conventional methods search from a single point, GAs always operate on a whole population of points (strings). This contributes to the robustness of genetic algorithms. It improves the chance of reaching the global optimum and, vice versa, reduces the risk of becoming trapped in a local stationary point.

3. Normal genetic algorithms do not use any auxiliary information about the objective function value such as derivatives.

4. GAs use probabilistic transition operators, conventional methods for continuous optimization apply exclusively deterministic transition operators.

From this point of view, it seems clear that genetic algorithms are robust methods which can, due to their generality, be applied to a wide range of different optimization problems. On the other hand, they can be of weak performance, because they disregard all information which can be useful.

Let us first consider the simpliest kind of a genetic algorithm which is widely used in the solution of continous optimization problems as well as in discrete optimization. It is characterized as a GA which operates on (not necessarily binary) strings of a fixed length n (see especially [Holland, 1992] and [Goldberg, 1989]).

3.1.2 Genetic Algorithms with Objects of Fixed Size

Above, we have already mentioned that GAs use probabilistic transition operators which are imitations of mechanisms occuring in the natural sexual reproduction process. A simple genetic algorithm, which operates on strings of fixed size, incorporates the following methods in its transition operator δ:

Selection: Individuals with high fitness are favored in the reproduction process. Concretely, individuals with high fitness have a higher probability to survive and to reproduce themselves.

Mutation: In real evolution the genetic material can by changed randomly by erroneous reproduction or other deformations of genes, e.g. by gamma radiation. In genetic algorithms mutation can be realized as a random deformation of the strings with a certain probability.

Crossing over: Method of merging the genetic information of two individuals; this mechanism has contributed much to the fast adaptation of sexually reproducing species.

We can summarize this in the following algorithm:

Algorithm 3.2 Let m be the size of the population.

$t := 0$;
Compute initial population $\mathcal{B}_0 = (b_{1,0}, \ldots, b_{m,0})$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
  FOR $i := 1$ **TO** m **DO**
  BEGIN
    select an individual g *from* $\mathcal{B}_t$;

    IF $\mathsf{Random}[0,1] \leq p_{\mathbf{C}}$ **THEN**
      cross g *with a randomly chosen individual of* $\mathcal{B}_t$;

    mutate g;

    $b_{i,t+1} := g$
  END

  $t := t + 1$
END

This algorithm already contains the mechanisms we mentioned above. Now it is time to take a closer look at these.

Selection

Selection is, in fact, the component which guides the algorithm to the solution. It should be a mechanism which favors high-fitted individuals but disadvantages low-fitted ones. It can be a deterministic operation, but, in most implementations, it has random components.

One variant, which is very popular nowadays, is the following scheme, where the probability to choose a certain individual is proportional to its fitness. It can be regarded as a random experiment with

$$\mathsf{P}[b_{j,t} \textit{ is selected}] := \frac{f(b_{j,t})}{\sum_{k=1}^{m} f(b_{k,t})}. \tag{3.5}$$

Of course, this formula only makes sense if all the fitness values are positive. If this is not the case, a transformation must be applied (a shift in the

simpliest case). This random experiment is, in some sense, a roulette game in which the probabilities to select a certain individual depends on its fitness.

As anticipated above, it can be necessary to use a transformed fitness function in the selection. Then the probabilities can be expressed as

$$\mathsf{P}[b_{j,t} \textit{ is selected}] := \frac{\varphi(f(b_{j,t}))}{\sum_{k=1}^{m} \varphi(f(b_{k,t}))} \tag{3.6}$$

where $\varphi : \mathbb{R} \to \mathbb{R}^+$ is a non-decreasing function. The function φ can also be useful to accelerate the accumulation of high-fitted individuals. Consider for instance $\varphi(x) := x^p$ with $p > 1$. In this case it becomes, depending on p, less probable to select low-fitted strings.

The algorithmical formulation of the selection scheme (3.5) can be written down as follows, analogously for the case of (3.6):

Algorithm 3.3

$x := \mathsf{Random}[0, 1]$;
$i := 1$

WHILE $i < m$ & $x < \sum_{j=1}^{i} f(b_{j,t}) / \sum_{j=1}^{m} f(b_{j,t})$ **DO**
    $i := i + 1$;

select $b_{i,t}$;

Summarized, this mathematical selection is an equivalent to the struggle for life in which high-fitted individuals have better chances to survive. This method will be called proportional selection in the following.

Crossing Over

In real sexual reproduction the genetic material of the two parents is merged during meiosis (see [Linder, 1979] or [Gerhardt *et al.*, 1976]). This mechanism is a very powerful tool to introduce new genetic material which is, with high probability, of higher fitness than its parents. Several investigations have shown that crossing over is the reason why sexually reproducing species have adapted faster than asexually reproducing ones.

Basically, crossing over is the exchange of genes between the chromosomes of the two parents. In real meiosis it is, simplistically, exchanging of parts of chromosomes. In our investigation of genetic algorithms with objects of fixed size crossing over can, in the simpliest case, be realized as

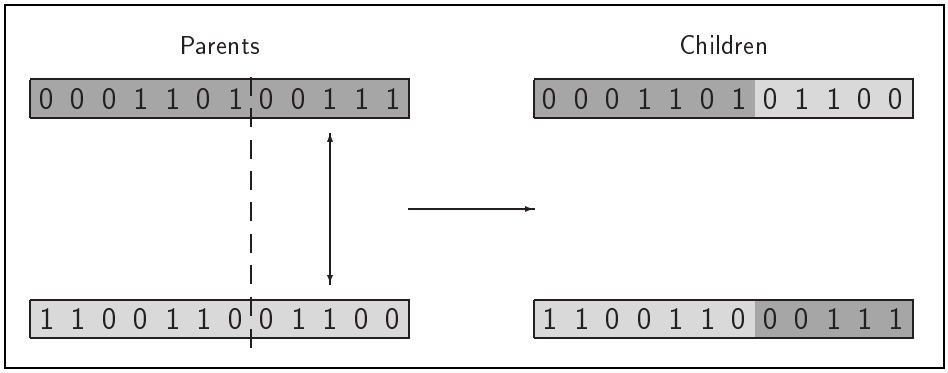


Figure 3.1: One-point crossing over of binary strings

cutting two strings at a randomly chosen position and swapping the two tails. This process is visualized in figure 3.1. We will call it one-point crossing over in the following.

Algorithm 3.4

$pos :=$ Random$\{1, \ldots, n-1\}$;

FOR $i := 1$ **TO** pos **DO**
BEGIN
    $Child_1[i] := Parent_1[i]$;
    $Child_2[i] := Parent_2[i]$;
END

FOR $i := pos + 1$ **TO** n **DO**
BEGIN
    $Child_1[i] := Parent_2[i]$;
    $Child_2[i] := Parent_1[i]$;
END

One-point crossing over is a simple and often-used method for GAs which operate on binary strings. For other problems, different codings, but also for the case of binary strings, different crossing over techniques are reasonable or can even be necessary. We mention just a few of them, for more details see [Geyer-Schulz, 1995]:

N-point crossing over: Instead of only one N breaking points are chosen randomly. Every second section is swapped.

Segmented crossing over: Similar to N-point crossing over with the difference that the number of breaking points can vary.

Uniform crossing over: For each position it is decided randomly if the positions are swapped.

Shuffle crossing over: First a randomly chosen permutation is applied to the two parents, then N-point crossing over is applied to the shuffled parents, finally, the shuffled children are transformed back with the inverse permutation.

Mutation

The last ingredient of our simple genetic algorithm is mutation. In the real world it is the random deformation of the genetic information of an individual by means of radioactive radiation or other outer influences. In real reproduction the probability that a certain gene is mutated is almost equal for all genes. So, it is near at hand to use the following mutation technique for a given binary string S:

Algorithm 3.5

FOR $i := 1$ **TO** n **DO**
  IF Random$[0,1] < p_{\mathrm{M}}$ **THEN**
    invert $S[i]$;

Again, similar to the case of crossing over, the choice of the appropriate mutation technique depends on the coding and the problem itself. We mention a few alternatives, more details can be found in [Geyer-Schulz, 1995]:

Inversion of single bits: With probability p_{M} one randomly chosen bit is negated.

Inversion: With probability p_{M} the whole string is inverted bitwise.

Random selection: With probability p_{M} the string is replaced by a randomly chosen one.

This completes our genetic algorithm. If we fill in the methods described above, we can write down a universal genetic algorithm for solving optimization problems in the space $\{0,1\}^n$.

Algorithm 3.6

$t := 0$;
Create initial population $\mathcal{B}_0 = (b_{1,0}, \ldots, b_{m,0})$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
  FOR $i := 1$ **TO** m **DO**
  BEGIN

(* proportional selection *)

    $x :=$ Random$[0, 1]$;
    $l := 1$
    WHILE $l < m$ & $x < \sum_{j=1}^{l} f(b_{j,t}) / \sum_{j=1}^{m} f(b_{j,t})$ **DO**
      $l := l + 1$;
    $g := b_{l,t}$;

(* one-point crossing over *)

    IF Random$[0, 1] \leq p_{\mathbf{C}}$ **THEN**
    BEGIN
      $Parent_1 := g$;
      $Parent_2 := b_{\text{Random}\{1,\ldots,m\},t}$;
      $pos :=$ Random$\{1, n-1\}$;

      FOR $i := 1$ **TO** pos **DO**
      BEGIN
        $Child_1[i] := Parent_1[i]$;
        $Child_2[i] := Parent_2[i]$;
      END
      FOR $i := pos + 1$ **TO** n **DO**
      BEGIN
        $Child_1[i] := Parent_2[i]$;
        $Child_2[i] := Parent_1[i]$;
      END

      IF Random$[0, 1] < \frac{1}{2}$ **THEN**
        $g := Child_1$;
      ELSE
        $g := Child_2$;
    END

(* mutation *)

    FOR $i := 1$ **TO** n **DO**
      IF Random$[0, 1] < p_{\mathbf{M}}$ **THEN**
        invert $g[i]$;
    $b_{i,t+1} := g$
  END

  $t := t + 1$;
END

In this implementation, as usual, n is the size of the strings and m is the size of the population.

3.1.3 Genetic Programming

The second important class of genetic algorithms is comprised under the term "genetic programming". It is a genetic approach to a very important problem in artificial intelligence — the problem of program induction. Program induction is the method of teaching a computer to solve a problem without being programmed explicitely. It is, in some sense, an inverse problem, instead of telling a computer explicitely how to solve a problem, the computer is shown the problem and encouraged to find a program which solves it. The idea of applying genetic algorithms to the problem of program induction can again be traced back to J. H. Holland. Very important research was also done by J. R. Koza who first introduced the term "genetic programming" (see [Koza, 1992]).

Obviously, the application of genetic algorithms to whole programs requires more or less significant modifications of the genetic machinery we discussed previously. The most important one is that we must get away from strings of fixed length. Although it is possible to restrict the length of a program to a certain value, we cannot preserve the universality of a programming language if we restrict the length of programs to a given number of statements. In the following we assume that the syntax of a programming language is given in Backus-Naur form (BNF) which describes the syntax of the language recursively. A program, which is written in such a language, can then be regarded as a (not necessarily binary) tree which can, of course, be written as a nested list. The process of rewriting a program as a tree or nested list can be done for every commonly used programming, language, such as, `C`, `FORTRAN`, `Pascal`, or `LISP`, to mention just a few. It is easy to see that it is least difficult for restricted languages which allow only recursions, such as `LISP`.

Now let us have a closer look at the basic things we need for an implementation of a genetic programming machinery. We do not go into very detail here, a detailed and comprehensive treatment of these things is provided e.g. in [Geyer-Schulz, 1995].

Choosing the Programming Language

As already mentioned above, it is possible to rewrite every program as a list or tree, but it can be a crucial task for most of the common procedural programming languages. In most applications and theoretical treatments `LISP` is used. `LISP` programs are nested lists themselves and, therefore, very easy to handle, because no complicated rewriting has to be done.

Another important aspect we must consider is which subset of the language we should use. It is clear that it is not useful in every application

to use the complete set of instructions the language offers. Consider for instance a problem where logical operations have to be learned. For such a case we can restrict to the logical operations AND, OR, and NOT, or even to a subset of these. Under the assumption that the problem can be solved within the subset, an intelligent choice of that subset can increase the performance remarkably. The simple reason is, that the search space is smaller. Consequently, this increases the chance to a considerably good solution.

Initialization

In the previous sections we did not pay any attention to the creation of the initial population. We assumed implicitely that the individuals of the first generation can be generated randomly with respect to a certain (mostly uniform) probability distribution. The random generation of trees or nested lists is a more subtle task.

Consider for instance the BNF of the following language which can be used for representing 3-ary logical functions:

```
      Program  :=  <expression> ;
<expression>   :=  "(" <variable> ")" |
                   "(" <unary> <expression> ")" |
                   "(" <binary> <expression> <expression> ")" ;
<variable>     :=  "x" | "y" | "z" ;
   <unary>     :=  "NOT" ;
  <binary>     :=  "AND" | "OR" ;
```

Obviously, the syntactical elements of this `LISP`-like language are parentheses, the variables x, y, and z, and the operators NOT, AND, and OR. It seems clear that a procedure, which creates strings with random entries from this set of syntactical elements, is not a good variant of an initialization procedure, because the probability of creating syntactically correct strings is rather low. A better alternative could be a procedure which is based on the BNF of the language itself. Such an algorithm can be outlined as follows:

1. Start with the root of the syntax (in our case Program)
2. Select an alternative of the current syntactical expression randomly
3. Fill in the alternative and apply the procedure recursively for all non-atomic subexpressions of the alternative

It is intuitively clear that we must include mechanisms which avoid endless recursion of this method. A common opportunity is to fix a maximum depth and to avoid non-atomic alternatives if this depth is reached or already

exceeded. The following example illustrates this method more clearly for the case of the language we defined above.

Example 3.7 Starting point is, of course, the root expression Program (the currently evaluated expression is marked by underlining).

$$\underline{\text{<expression>}}$$

second alternative

$$(\underline{\text{<unary>}}\ \text{<expression>}\ \text{<expression>})$$

<unary> is an atomic expression with only one alternative

$$(\text{NOT}\ \underline{\text{<expression>}}\ \text{<expression>})$$

third alternative

$$(\text{NOT}\ (\underline{\text{<binary>}}\ \text{<expression>}\ \text{<expression>})\ \text{<expression>})$$

first alternative

$$(\text{NOT}\ (\text{AND}\ \underline{\text{<expression>}}\ \text{<expression>})\ \text{<expression>})$$

first alternative

$$(\text{NOT}\ (\text{AND}\ (\underline{\text{<variable>}})\ \text{<expression>})\ \text{<expression>})$$

second alternative

$$(\text{NOT}\ (\text{AND}\ (y)\ \underline{\text{<expression>}})\ \text{<expression>})$$

... and so on.

Crossing Programs

The second thing, which differs from case of strings significantly, is the crossing over operation. Of course, nested lists can be regarded as strings to which a standard crossing over operation can be applied. The problem is again that the probability of generating syntactically correct strings this way is very low. Hence, it is necessary to find an alternative which preserves syntactical correctness.

For this purpose, we use the interpretation of programs as trees. Of course, there are many possible interpretations of this kind, the approach we discuss here was suggested by A. Geyer-Schulz and can be found in [Geyer-Schulz, 1995]. A correct program must be derivable from the underlying grammar in at least one (not necessarily unique) way. This derivation tree itself is then used as a representation. It is not so trivial to give an exact formulation of the procedure of determining the derivation tree for a given expression. We give a simple example which should make everything clear.

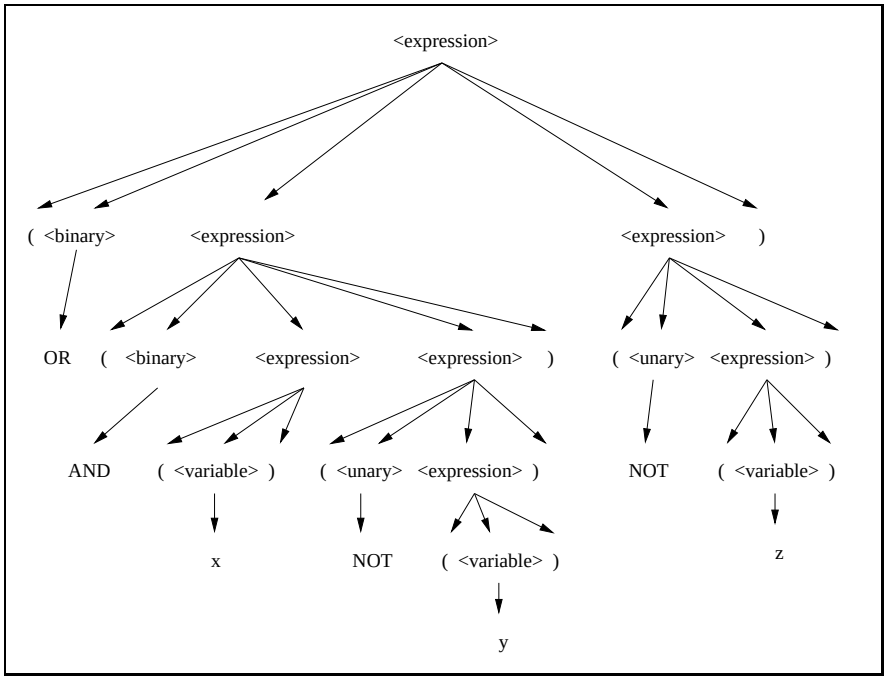


Figure 3.2: The derivation tree for (OR (AND (x) (NOT (y))) (NOT (z)))

Example 3.8 Figure 3.2 shows the derivation tree of the expression

$$((x \wedge \neg y) \vee \neg z)$$

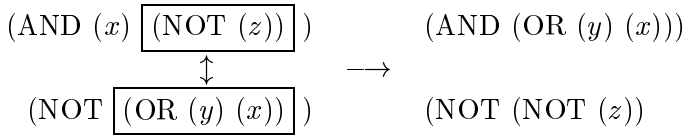
which can be written as

(OR (AND (x) (NOT (y))) (NOT (z)))

in our language.

It can be seen easily that every subtree corresponds to a subexpression. The roots of these subtrees are called labels. The most common method for crossing two expressions, which can be expressed with derivation trees, is now to exchange subtrees which start with equal nodes, i.e. which are equally labeled. This guarantees that the children are again syntactically correct.

Example 3.9 Figure 3.3 shows a simple example for crossing two derivation trees. The result in the form of nested lists is the following:



Mutating Programs

After all this preparatory work it is comparatively easy to give a mutation technique for programs of the usual form. The most common method is to select a subtree of the derivation tree randomly and to replace it by another subtree which was generated randomly by applying the same method we have discussed in connection with the initialization procedure. Of course, we have to pay special attention to the depth of this subtree again. Then it is guaranteed that the syntactical correctness of the program is not damaged by the mutation operation.

The Fitness Function

Another nontrivial task in connection with genetic programming is the definition of an appropriate fitness function which measures how good a given program solves the problem. The solution of this problem depends strongly on the problem itself, a universal recipe for defining the proper fitness measure cannot be given seriously. One commonly used technique is to apply a program to a finite number of test inputs for which the desired output is known. Of course, these cases must be selected in a careful way such that they are really representative. Then, for instance, the number of cases for which the correct output is obtained can be taken as a measure for the correctness of a program. The following example shows a case where this method is not useful.

Example 3.10 Let us consider the following grammar:

Function	:=	<expression> ;
<expression>	:=	"(" <variable> ")" |
		"(" <constant> ")" |
		"(" <binary> <expression> <expression> ")" ;
<variable>	:=	"x" ;
<constant>	:=	<number>"/"<number> ;
<number>	:=	"0" | $\cdots$ | "9" | <number> ;
<binary>	:=	"+" | "-" | "$*$";

This grammar describes a programming language for representing polynomials with rational coefficients. Our task is now to find a polynomial of some

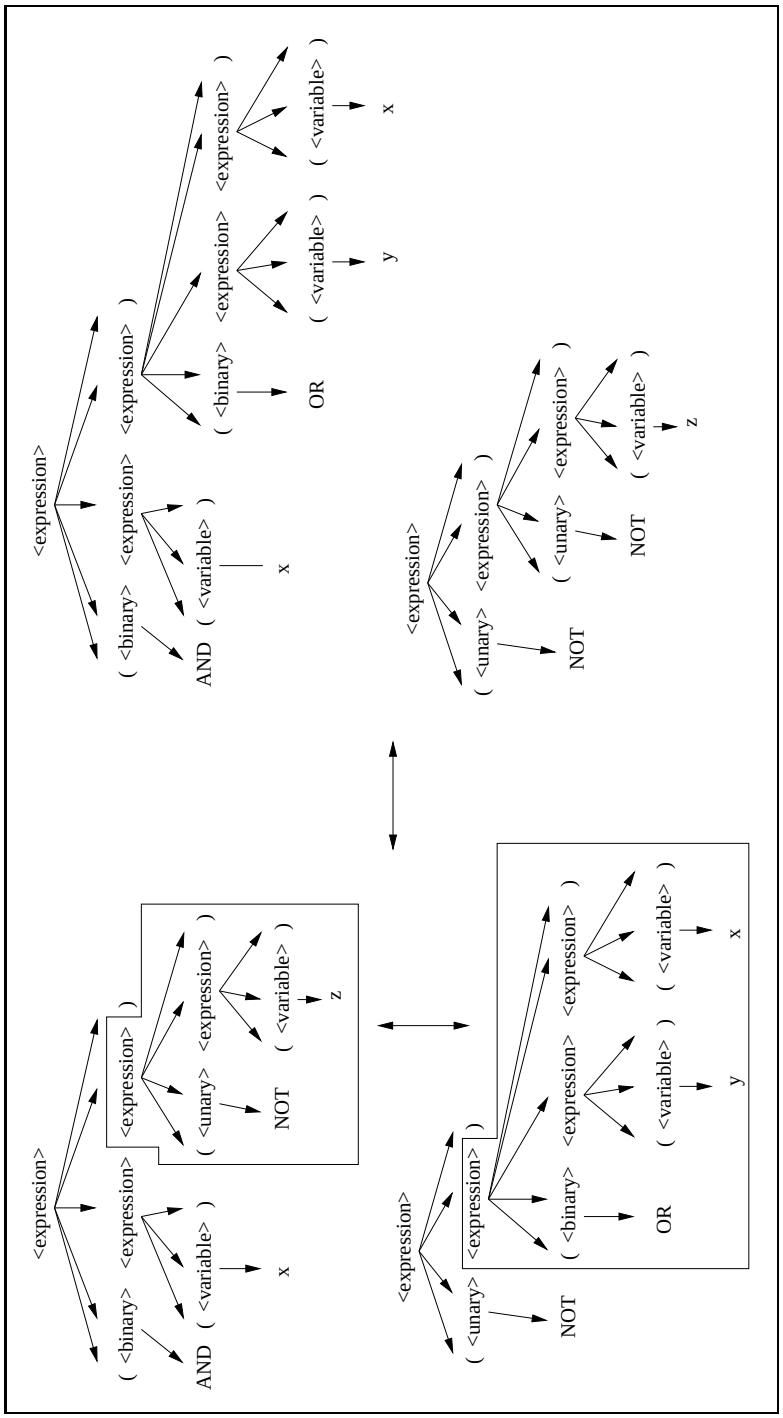


Figure 3.3: Example for crossing two derivation trees

bounded degree such that it fits a given set of representative test points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ as good as possible. Since the probability that a randomly chosen polynomial fits at least one point exactly is nearly 0, the number of exact matches are useless as fitness measure. An alternative would be

$$f(P) := \|(y_1 - P(x_1), \ldots, y_n - P(x_n))\|,$$

which should be minimized, where f is the fitness function of P, a given polynomial, and $\|.\|$ is an arbitrary (e.g. the Euclidean) norm on $\mathbb{R}^n$.

Summary

Assume that we have a program induction problem, the programming language is chosen, its syntax can be described by a BNF, we have a set of representative test cases, and we have a function, which measures the matching between the actually obtained and the desired output, then we can write down a genetic algorithm for solving the program induction problem. This can be done easily by replacing the strings by programs and the initialization-, crossing over-, and mutation techniques in 3.2 by the adapted operations we have discussed here.

In general, genetic programming can be applied to a wide range of different problems such as optimal control, symbolic regression, sequence induction, equation solving, discovering game playing strategies, inverse kinematics, or decision tree induction. More examples and applications can be found in [Koza, 1992] and [Geyer-Schulz, 1995] to which we also refer for more information about genetic programming, especially theoretical considerations, such as convergence theory.

3.2 Convergence Theory

After this practical survey of evolutionary computation, it is time to take a close look at the theoretical properties of such methods. For conventional deterministic optimization methods, such as gradient methods, Newton- or Quasi-Newton methods, it is usual to have results which guarantee that the sequence of iterations converges to a local optimum, more exactly to a stationary point, with a certain speed or order. For any probabilistic optimization method theorems of this kind cannot be given, because the behavior of the algorithm is not determinable in general. So, statements about the convergence of probabilistic optimization methods can only give information about the expected or average behavior of such algorithms. For the case of genetic algorithms there are a few circumstances which make it even more difficult to examine their convergence:

- Since a single transition from one generation to the next one is a combination of usually three probabilistic operators (selection, crossing over, and mutation), the inner structure of a genetic algorithm is rather complicated, which increases the difficulty of examining the convergence badly.
- For each of the involved probabilistic operators many different variants are known, thus it is not possible to give general convergence results due to the fact that the choice of the operators influences the convergence fundamentally.

In this section some important convergence results are provided. Unfortunately, they do not have the significance as some results in conventional continous optimization do. For simplicity, we restrict to the case of GAs with a fixed number m of objects of fixed size n.

The first little theorem states that the convergence of a certain variant is bounded below by the convergence of random search methods. It can be regarded as boundary for the convergence in the worst case.

Theorem 3.11 *Consider a genetic algorithm of type 3.2 where random selection with uniform distribution is used as mutation technique (see page 39). Then the probability to reach the global optimum is* 1. *The expected number of generations it takes to reach this optimum is at most*

$$\frac{1}{1-\left(1-p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}\right)^m}, \tag{3.7}$$

where $N_{\mathbf{opt}}$ *is the number of strings which have maximal fitness.*

Proof: In each step of a reproduction process the probability to reach the global optimum is at least

$$p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}.$$

Then the probability not to find the optimum in one complete reproduction step is at most

$$\left(1-p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}\right)^m.$$

The probability of finding the maximum at all is then

$$1-\lim_{k\to\infty}\left(\left(1-p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}\right)^m\right)^k=1-0=1.$$

The probability to find the global maximum at first in the k-th generation is at least

$$\left(\left(1-p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}\right)^m\right)^{k-1}\cdot\left(1-\left(1-p_{\mathbf{M}}\cdot\frac{N_{\mathbf{opt}}}{2^n}\right)^m\right).$$

Then the expected number of generations it takes until the optimum is reached is at most

$$\sum_{k=1}^{\infty} k \cdot \left(\left(1 - p_{\mathbf{M}} \cdot \frac{N_{\mathbf{opt}}}{2^n}\right)^m\right)^{k-1} \cdot \left(1 - \left(1 - p_{\mathbf{M}} \cdot \frac{N_{\mathbf{opt}}}{2^n}\right)^m\right) \overset{(*)}{=} \frac{1}{1-\left(1-p_{\mathbf{M}} \cdot \frac{N_{\mathbf{opt}}}{2^n}\right)^m},$$

what completes the proof. The identity $(*)$ can be derived easily from the Taylor series of $\frac{1}{x^2}$ at $x_0 = 1$. ■

Remark 3.12 This expected number is very high even for moderate problems. Therefore, this result is of no practical relevance at all.

Before we turn to the next theorem we need a few prerequisites.

Definition 3.13

1. A string $H = (h_1, \ldots, h_n)$ over the alphabet $\{0, 1, *\}$ is called a schema. An $h_i \neq *$ is called a specification, an $h_i = *$ is called a wildcard. From the definition of the injective mapping

$$\begin{array}{rccl} i: & \{0,1,*\}^n & \to & P(\{0,1\}^n) \\ & H & \mapsto & i(H) := \{G | \forall i \in \overline{1,n}: \ h_i \neq * \Rightarrow h_i = g_i\} \end{array}$$

   it is clear that schemata can be regarded as special subsets of $\{0,1\}^n$.

2. A string $G = (g_1, \ldots, g_n)$ over the alphabet $\{0, 1\}$ fulfills the schema $H = (h_1, \ldots, h_n) \quad :\Longleftrightarrow \quad \forall i \in \{j | h_j \neq *\} : g_i = h_i$. We denote that with $G \in H$.

3. $\mathcal{O}(H) := |\{i \in \{1, \ldots, n\} | h_i \neq *\}|$ is called the order of the schema H (number of specifications).

4. The distance between the first and the last specification $\delta(H) := \overline{i} - \underline{i}$, where $\underline{i} := \min\{i | h_i \neq *\}$ and $\overline{i} := \max\{i | h_i \neq *\}$, is called the defining length of the schema H.

The following theorem is *the* fundamental theorem on the convergence of GAs. The main statement is that schemata with a higher-than-average fitness accumulate geometrically if proportional selection is used.

Theorem 3.14 (Schema Theorem - Holland 1975)
Consider a genetic algorithm of type 3.2 with proportional selection. Then the inequality

$$\mathsf{E}[r_{H,t+1}] \geq \frac{\tilde{f}(H,t)}{\bar{f}(t)} \cdot r_{H,t} \cdot P_C(H) \cdot P_M(H) \tag{3.8}$$

holds with the notations

$r_{H,t}$ ... *number of individuals which fulfill the schema H at time t*

$\bar{f}(t)$ ... *average fitness of population at time t*

$\tilde{f}(H,t)$ ... *estimated average fitness of schema H at time t, $\tilde{f}(H,t) := \frac{1}{r_{H,t}} \sum_{b_{i,t} \in H} f(b_{i,t})$*

$P_C(H)$ ... *constant which depends only on the schema H and the crossing over method*

$P_M(H)$ ... *constant which depends only on the schema H and the mutation operator*

Depending on the involved methods, we can specify the constants $P_C(H)$ and $P_M(H)$ in the following ways:

$P_C(H) := 1 - p_{\mathbf{C}} \cdot \frac{\delta(H)}{n-1}$	*for one-point crossing over*
$P_C(H) := 1 - p_{\mathbf{C}} \cdot \left(1 - \left(\frac{1}{2}\right)^{\mathcal{O}(H)}\right)$	*for uniform crossing over*
$P_C(H) := 1 - p_{\mathbf{C}}$	*for any other crossing over method*
$P_M(H) := (1 - p_{\mathbf{M}})^{\mathcal{O}(H)}$	*for normal bitwise mutation*
$P_M(H) := 1 - p_{\mathbf{M}} \cdot \frac{\mathcal{O}(H)}{n}$	*for the inversion of a single bit*
$P_M(H) := 1 - p_{\mathbf{M}} \cdot \frac{\lvert H \rvert}{2^n}$	*for random selection*

Proof: Let the schema H be arbitrary but fixed. Assume that we have to perform the transition from generation $\mathcal{B}_t$ to generation $\mathcal{B}_{t+1}$. Then the number $r_{H,t}$ of individuals which fulfill H at time t is known. The number $r_{H,t+1}$ of individuals which fulfill H at time $t+1$ is determined by the probabilistic operators and, therefore, a random variable. We can write $r_{H,t+1}$ as

$$r_{H,t+1} = \sum_{i=1}^{m} Z_i$$

where Z_i are the following random variables

$$Z_i := \begin{cases} 1 & \text{if } b_{i,t+1} \text{ fulfills } H, \\ 0 & \text{otherwise.} \end{cases}$$

It can be easily seen from algorithm that the random variables Z_i are completely independent and distributed equally. Hence, we can write

$$\mathsf{E}[r_{H,t+1}] = m \cdot \mathsf{E}[Z], \tag{3.9}$$

where Z is the random variable

$$Z := \begin{cases} 1 & \text{an individual selected from } \mathcal{B}_t, \text{ crossed with a randomly chosen individual, and mutated afterwards, fulfills } H, \\ 0 & \text{otherwise.} \end{cases}$$

The expectation of Z can then be estimated as follows (use theorem A.14 which delivers the multiplicativity of the expectation):

$$\mathsf{E}[Z] \geq p_1 \cdot p_2 \cdot p_3, \tag{3.10}$$

where p_1 is the probability that an individual, which fulfills H, is selected, p_2 is the probability that an individual, which fulfills H, still fulfills H after it is crossed with an arbitrary one, and p_3 is the probability that an individual, which fulfills H, still fulfills H after it is mutated. The probability to select an individual from $\mathcal{B}_t$, which fulfills H, is, if proportional selection is used,

$$p_1 = \frac{\sum\limits_{b_{i,t} \in H} f(b_{i,t})}{\sum\limits_{i=1}^{m} f(b_{i,t})}. \tag{3.11}$$

If we join (3.9), (3.10), and (3.11) together, and if we fill in the lower boundaries for p_2 and r_3, $P_C(H)$ and $P_M(H)$, respectively, we get the result

$$\begin{aligned}
\mathsf{E}[r_{H,t+1}] &= m \cdot \mathsf{E}[Z] \geq m \cdot p_1 \cdot p_2 \cdot p_3 \geq m \cdot \frac{\sum\limits_{b_{i,t} \in H} f(b_{i,t})}{\sum\limits_{i=1}^{m} f(b_{i,t})} \cdot P_C(H) \cdot P_M(H) \\
&= \frac{\frac{\sum\limits_{b_{i,t} \in H} f(b_{i,t})}{r_{H,t}}}{\frac{\sum\limits_{i=1}^{m} f(b_{i,t})}{m}} \cdot r_{H,t} \cdot P_C(H) \cdot P_M(H) = \frac{\tilde{f}(H,t)}{\bar{f}(t)} \cdot P_C(H) \cdot P_M(H).
\end{aligned}$$

Now it remains to show the lower boundaries of the probabilities p_2 and p_3, $P_C(H)$ and $P_M(H)$, respectively.

- Obviously, the probability that an individual still fulfills H after the crossing over step is at least $1 - p_{\mathbf{C}}$, because crossing over is only performed with a probability $p_{\mathbf{C}}$.
- For the case of one-point crossing over we can conclude further that the fulfillment of schema H is only destroyed if the breaking point lies within the defining length of the schema. The probability that the breaking point lies within the defining length is $\frac{\delta(H)}{n-1}$. For this case we can define $P_C(H) := 1 - p_{\mathbf{C}} \cdot \frac{\delta(H)}{n-1}$.
- The probability that schema H is disrupted by uniform crossing over is $1 - \left(\frac{1}{2}\right)^{\mathcal{O}(H)}$. So, the probability that an individual, which fulfills H, still fulfills H after uniform crossing over is at least $1 - p_{\mathbf{C}} \cdot \left(1 - \left(\frac{1}{2}\right)^{\mathcal{O}(H)}\right)$.
- The probability that the fulfillment of schema H is not damaged by normal bitwise mutation is, of course, $(1 - p_{\mathbf{M}})^{\mathcal{O}(H)}$.

- The probability that a specification of H is chosen by the inversion-of-one-single-bit mutation is exactly $\frac{\mathcal{O}(H)}{n}$. Therefore the probability that an individual still fulfills H after this kind of mutation is $1 - p_{\mathbf{M}} \cdot \frac{\mathcal{O}(H)}{n}$.
- The probability that an individual, which fulfills H, is chosen by uniform random selection is $\frac{|H|}{2^n}$. Thus, the probability that the mutated individual also fulfills H is $1 - p_{\mathbf{M}} \cdot \frac{|H|}{2^n}$.

This completes the proof. ■

From the schema theorem we can deduce the following generalization directly:

Corollary 3.15 *With the notations and assumptions of theorem 3.14 the inequality*

$$\mathsf{E}[r_{H,t+k}] \geq r_{H,t} \cdot P_C(H)^k \cdot P_M(H)^k \cdot \prod_{i=0}^{k-1} \frac{\tilde{f}(H, t+i)}{\bar{f}(t+i)} \tag{3.12}$$

holds for every $k \geq 1$.

Proof: Follows directly by applying theorem 3.14 k times (see also [Holland, 1992]). ■

It is easy to see from the last corollary that schemata with a higher-than-average fitness accumulate exponentially, especially if they have a low order and, if one-point crossing over is used, a short defining length. Such schemata are called building blocks. From these considerations it might be clear that, in some sense, convergence is only possible if building blocks exist at all. This actually means that the coding must be chosen in a way such that some certain features, i.e. schemata, correspond to a high fitness. Then the probability of obtaining even better fitted children by crossing good parents is higher than if this is not the case. The strategy of accumulating high-fitted schemata without forgetting completely about the other ones is called implicit parallelism and contributes much to the robustness of genetic algorithms and the chance to reach the global solution. For a detailed mathematical investigation of the phenomenon of implicit parallelism we refer to [Goldberg, 1989], [Geyer-Schulz, 1995], and, in particular, [Holland, 1992].

Summarized, the convergence of genetic algorithms depends strongly on how careful the coding was chosen. This must be done in a way that there are building blocks. Unfortunately, there is not a more reliable convergence theory for GAs of this general kind.

3.3 Extensions, Generalizations

Finally, we introduce concepts which differ from the simple constructs we have discussed previously.

Other Selection Schemes

Instead of proportional selection, many other selection methods are known and successfully applied. We mention just a few:

Linear rank selection: Instead of the normalized fitness the rank of the fitness in the generation is used as basis of the selection.

Tournament selection: A group of individuals is sampled from the population, the individual with best fitness is chosen for reproduction.

Elitism: It can often be of advantage to avoid that the best-fitted individual dies out. This can be achieved easily by selecting the best individuals always without taking the other circumstances into account. This method has often been used successfully in addition to other selection methods.

Adaptive GAs

Adaptive genetic algorithms are GAs whose parameters, such as the population size, the crossing over probability, or the mutation probability, or even the genetic operators for selection, crossing over, and mutation, are varied with the time (e.g. see [Chen and Chang, 1993]).

Hybrid GAs

As they use the fitness function only in the selection step, genetic algorithms are blind optimizers which do not use any auxiliary information such as derivatives or other specific knowledge about the special structure of the function. If there is such knowledge, it is possible to incorporate other optimization techniques in order to support the GA and, consequently, to improve convergence. This can, on the one hand, improve the speed considerably, but it is, on the other hand, possible that the risk of premature convergence also increases.

Self Organizing GAs

In the real world, the reproduction methods themselves and the representations of the genetic material were adapted through the billions of years of evolution (see [Rechenberg, 1973] for reference). Many of these adaptations of the natural genetic algorithm were able to increase the speed of adaptation of the individuals. It is reasonable to encode not only the raw genetic information but also further information, for example parameters of the coding function. This can lead to an automated adaptation of the coding such that more significant building blocks can be accumulated, which increases the speed of adaptation, i.e. the convergence.

Chapter 4

Optimizing Fuzzy Sets with Genetic Algorithms

Once upon a time a fire broke out in a hotel, where just then a scientific conference was held. It was night and all guests were sound asleep. As it happened, the conference was attended by researchers from a variety of disciplines. The first to be awakened by the smoke was a mathematician. His first reaction was to run immediately to the bathroom, where, seeing that there was still water running from the tap, he exclaimed: "There is a solution!". At the same time, however, the physicist went to see the fire, took a good look and went back to his room to get an amount of water, which would be just sufficient to extinguish the fire. The electronic engineer was not so choosy and started to throw buckets and buckets of water on the fire. Finally, when the biologist awoke, he said to himself: "The fittest will survive" and went back to sleep.

Anecdote originally told by C. L. Liu

As already mentioned, fuzzy systems are capable of performing human-like decisions by processing imprecise knowledge in a rule-based way. Since both fuzzy sets and approximate reasoning are only models of the representation of imprecise information and the inference method humans use, the way of translating human knowledge into a fuzzy system is not a straightforward one.

We can state that the behavior of a fuzzy system depends on three disjoint sets of parameters. First of all, the fuzzy subsets of the corresponding

universes, which are associated with the verbal values of the linguistic variables, determine the semantics of the rules. Second, the rules themselves determine how the output is computed for a given input. Last, the involved operations, such as t-norms and t-conorms, linguistic approximation, or defuzzification, influence the shape of the output surface too.

Many researchers have reported that they were able to design a prototype of a fuzzy system for a given task rapidly, but that it took a lot of time and work to tune all these parameters. Since the complexity of the problems, to which fuzzy systems are applied, is increasing strongly, it is recommendable or even indispensable to take a closer look at methods for tuning the parameters of a fuzzy system automatically.

If we assume that the structure of the fuzzy system, the universes of discourse, and the operations are given, we can distinguish between three different learning tasks:

1. The rules are given, but the fuzzy sets are unknown at all and must be found or, what happens more often, they can only be estimated und must be optimized. A typical example would be the following: The rules for driving a car are teached in a driving school (e.g. "For starting a car let in the clutch slowly and, simultaneously, step on the gas carefully."), but the beginner must learn from practical experience what "slowly letting in the clutch" actually means.

2. The semantical interpretation of the rules is, at least sufficiently well known, but the relations between the input and the output, i.e. the rules, are unknown. A typical example is extracting certain risk factors from patient data. In this case, it is sufficiently known which blood pressures are high and which are low, but the factors, which really influence the risk of a certain disease, are unknown.

3. Nothing is known, both fuzzy sets and rules must be acquired, for instance from sample data.

The acquisation of rules is discussed in the next chapter. The third task will then be treated briefly in chapter 6. The first one is the subject of this chapter, where 4.1 provides some theoretical aspects of the application of GAs to the optimization of fuzzy sets and 4.2 demonstrates practical results.

Typically, in the case of the first problem, we get constrained optimization problems in spaces of uncountable size. If the fitness function, which judges the performance of the fuzzy system depending on the configuration of the fuzzy sets, is sufficiently smooth, it is possible to apply conventional optimization method which are, in almost all cases, faster than GAs. The reason why genetic algorithms are used, although they are disadvantaged

compared with conventional methods, is, on the one hand, that the fitness function is often not as smooth as necessary, and, on the other hand, that genetic algorithms often offer a better chance to reach the optimal solution instead of becoming trapped in a local stationary point. One aspect, which should not be neglected either, is that GAs are rather easy to implement.

4.1 Fuzzy GAs

A fuzzy genetic algorithm, we will abbreviate this with "fuzzy GA" in the following, is a genetic algorithm which is applied to the optimization of some parameters of a fuzzy system.

The first thing, which must be examined, is how to encode fuzzy sets or, more exactly, the parameters, which describe them, into binary strings.

4.1.1 Coding Fuzzy Sets

Here, our main aim is to represent a whole configuration of fuzzy sets by a binary string. As already mentioned, we assume that the rules and the operators are fixed and, consequently, that only the fuzzy sets can be varied. The number of rules is finite, hence the number of fuzzy subsets of corresponding universes, which are to be considered, is also finite. This collection of fuzzy sets consists of all the semantical interpretations $M(v)$ of those expressions v, which occur in at least one rule. Assume that we have already strings, which represent each one of the fuzzy sets, we can concatenate them to one string which is then a representation of the whole configuration. Hence, the only problem, which remains to be solved, is how to encode single fuzzy sets.

Since $\mathcal{F}(X)$ is an uncountable set even if X is finite, we must restrict to certain subsets of $\mathcal{F}(X)$, if we want to represent them by a binary string of finite length.

The Case of a Finite Universe of Discourse

If the corresponding universe of discourse X is finite, a fuzzy subset of X can be regarded as a vector of dimension $n := |X|$ with entries from $[0, 1]$. Assume that we have a coding, i.e. a discretization of real values between 0 and 1 of length m, then an arbitrary fuzzy subset of X can be encoded with

$$\begin{array}{rccl} c: & \mathcal{F}(X) \simeq [0,1]^n & \longrightarrow & \{0,1\}^{n \cdot m} \\ & (x_0, \ldots, x_{n-1}) & \longmapsto & (b_0, \ldots, b_{n \cdot m - 1}), \end{array} \tag{4.1}$$

where the substring $(b_{i \cdot m}, \ldots, b_{(i+1) \cdot m - 1})$ is the coded version of x_i.

Example 4.1 A typical method of encoding a value of a finite real interval $[a, b]$ is mapping a value x to a whole number between 0 and $2^m - 1$ by

$$\begin{array}{rccl} c_{m,[a,b]}: & [a,b] & \longrightarrow & \{0,\ldots,2^m-1\} \\ & x & \longmapsto & \left\lfloor (2^m-1)\cdot\frac{x-a}{b-a}\right\rfloor, \end{array} \tag{4.2}$$

which can then be encoded into a binary string by a transformation to its dual representation. The corresponding decoding can be done by computing the decimal representation K of the string, which ranges, of course, between 0 and $2^m - 1$, and by applying the following transformation to K:

$$\begin{array}{rccl} \tilde{c}_{m,[a,b]}: & \{0,\ldots,2^m-1\} & \longrightarrow & [a,b] \\ & K & \longmapsto & a + K\cdot\frac{b-a}{2^m-1}. \end{array} \tag{4.3}$$

It is easy to see that $\tilde{c}$ is injective and that $(c \circ \tilde{c}) \equiv \mathrm{id}_{\overline{0,2^m-1}}$.

In the following we will assume implicitely, if not indicated otherwise, that real values are encoded this way. Another possibility, if the values should not be bounded, is to use the binary floating point representation of a real number.

Coding Fuzzy Subsets of a Real Interval $[a, b]$

In this section some important classes of fuzzy subsets of a finite real interval, which can easily be encoded, are discussed. The case of real intervals as universes of discourse is very important, especially in control applications.

Starting points of the first class of fuzzy subsets of a real interval, which we want to discuss here, are a given discretization ($a = x_0, x_1, \ldots, x_{n-1}, x_n = b$), which is, in the simpliest case, a partition with equally sized parts, and an interpolation technique, linear interpolation in one of the simpliest cases. Then all the fuzzy subsets, which are given as interpolation of the membership degrees in the grid points x_i, are uniquely determined by these membership values. Figure 4.1 shows a typical fuzzy set of this kind. By the way, this is the method how fuzzy sets are represented in popular fuzzy control tools, such as *fuzzy*TECH or TILShell.

Analogously to the coding of fuzzy subsets of finite sets, the membership values at the grid points can be encoded into a binary string, e.g. with the method proposed in example 4.1.

The next two classes are, in some sense, a subset of the previous one. They are special kinds of piecewise linear membership functions. Figure 4.2 shows a so-called triangular membership function. It can be seen easily that its shape depends on three parameters — the value r where its modus is lying, a left offset u, and a right offset v. A membership functions of this kind can then easily be encoded by

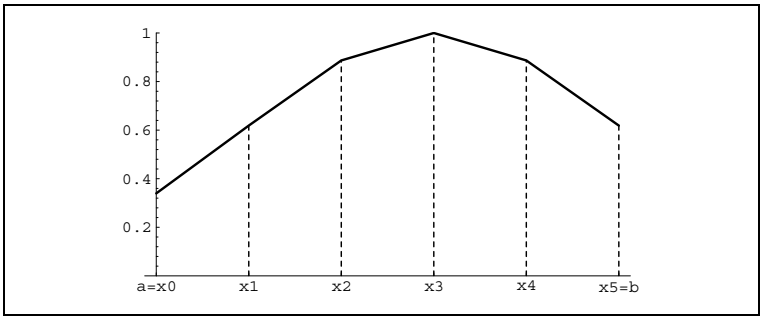


Figure 4.1: A fuzzy subset which is given by the membership values in a finite number of grid points

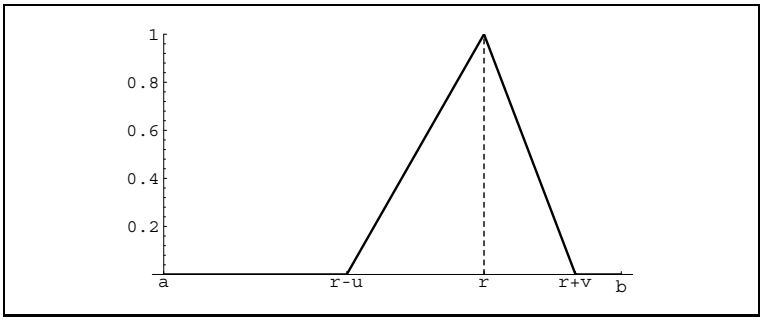


Figure 4.2: A triangular fuzzy set

$c_{m,[a,b]}(r)$	$c_{m,[0,\delta]}(u)$	$c_{m,[0,\delta]}(v)$

,

where δ is an upper boundary for the size of the offsets, for example $\delta := (b-a)/2$. Codings of this kind can be found in the 1993 papers of M. A. Lee and H. Takagi ([Lee and Takagi, 1993a], [Lee and Takagi, 1993b], [Lee and Takagi, 1993c], and [Takagi and Lee, 1993]).

A more general case, which is also widely used, are the so-called trapezoid fuzzy sets. As apparent from figure 4.3, the shape of a trapezoid membership function depends on four values, a value r, where the interval of maximal membership 1 starts, a value q, the length of this interval, and again a left offset u and a right offset v. Analogously, a coding can be defined as

$c_{m,[a,b]}(r)$	$c_{m,[0,\delta]}(q)$	$c_{m,[0,\delta]}(u)$	$c_{m,[0,\delta]}(v)$

.

In some control applications, where the smoothness of the control surface plays an important role, fuzzy sets of higher differentiability must be used.

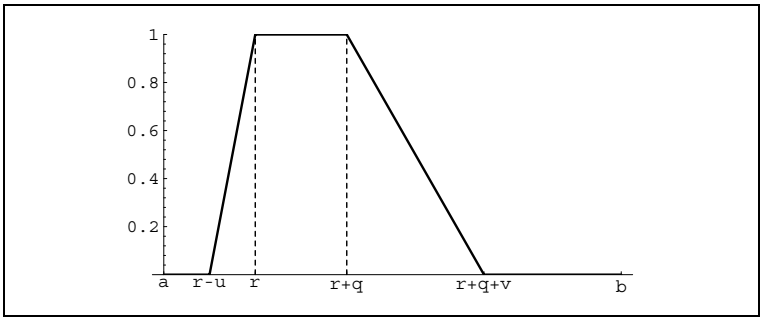


Figure 4.3: A trapezoid fuzzy set

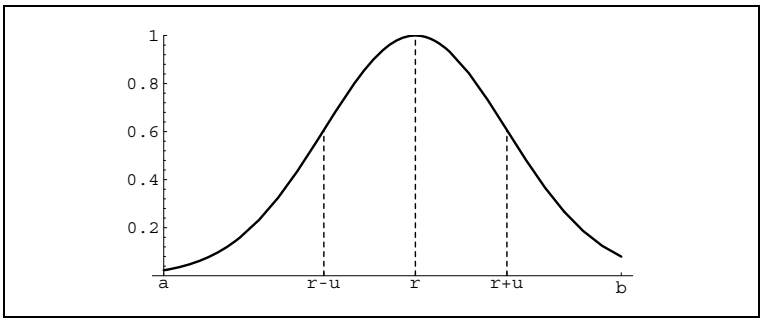


Figure 4.4: Typical bell-shaped fuzzy set

The most prominent representative of such a fuzzy set is the bell-shaped fuzzy set whose membership function is given by a Gaussian bell function:

$$x \longmapsto e^{-\frac{(x-r)^2}{2u^2}} \tag{4.4}$$

Figure 4.4 shows a typical membership function of this kind. Obviously,

$c_{m,[a,b]}(r)$	$c_{m,[\varepsilon,\delta]}(u)$

is an appropriate coding for the two parameters r and u which describe the shape of a bell-shaped fuzzy set, where ε is a lower boundary for the offset. Sometimes, the membership function is additionally scaled with a height factor $h \in (0, 1]$ whose coding must be appended to the string above (cf. [Furuhashi *et al.*, 1995]).

The "bell-shaped analogon" to trapezoid fuzzy sets are the so-called radial basis functions (see [Shimojima *et al.*, 1995]):

$$x \longmapsto \begin{cases} e^{-\frac{(|x-r|-q)^2}{2u^2}} & \text{if } |x-r| > q \\ 1 & \text{if } |x-r| \leq q \end{cases} \tag{4.5}$$

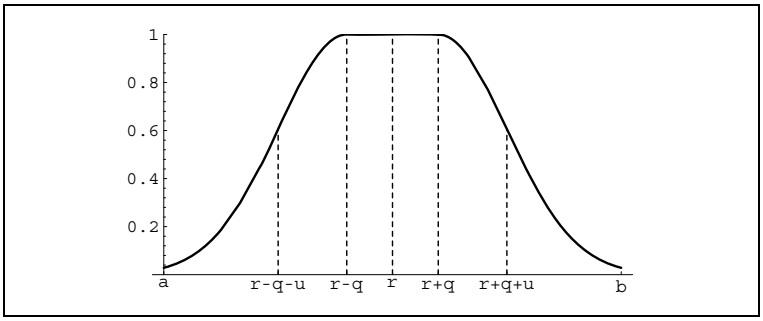


Figure 4.5: Fuzzy set with radial basis membership function

From figure 4.5, which shows a typical representative, it can be seen that r is the middle of the fuzzy set, q is the radius of the interval of maximal degree of membership 1 (the "basis"), and u is again the offset, the width of the bell. An appropriate coding would be the following:

$c_{m,[a,b]}(r)$	$c_{m,[0,\delta]}(q)$	$c_{m,[\varepsilon,\delta]}(u)$

4.1.2 Coding Whole Fuzzy Partitions

In many applications there is some a priori knowledge about the approximate configuration, for instance, something like an ordering of the fuzzy sets. In this case a general coding of each fuzzy subset of one linguistic variable would neglect this knowledge and, consequently, yield an unnecessarily large search space, which could be considerably smaller if some degrees of freedom were removed.

Example 4.2 A typical situation, not only in control applications, is that we have a certain number of fuzzy sets with labels, such as "negative big", "negative medium", "negative small", "approximately zero", "positive small", "positive medium", and "positive big". In such a case we have a natural ordering of the fuzzy sets. Obviously, an arbitrary configuration of the fuzzy sets would be senseless.

An often-used construct for representing fuzzy sets of linguistic variable of such a type are the so-called fuzzy partitions.

Definition 4.3 A finite family $(A_1, \ldots, A_n)$ of fuzzy subsets of a domain X is called a fuzzy partition if and only if

$$\forall x \in X : \sum_{i=1}^{n} \mu_{A_i}(x) = 1. \tag{4.6}$$

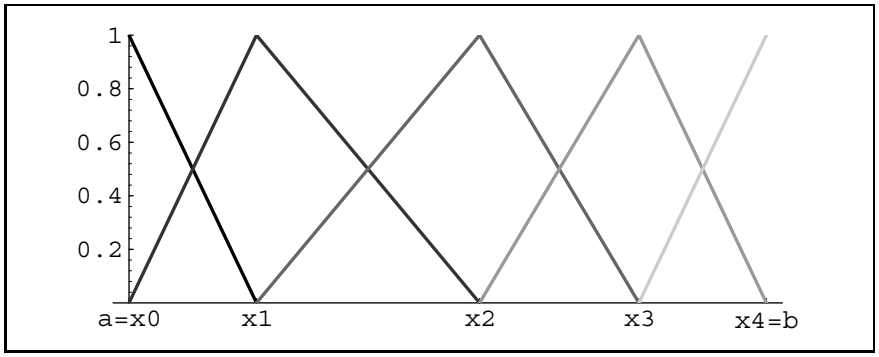


Figure 4.6: A typical fuzzy partition with $n = 5$ triangular parts

This definition goes back to E. H. Ruspini (cf. [Ruspini, 1969]) and is the notion of a fuzzy partition which is mostly used in practical applications. Of course, other generalizations of classical crisp partitions are reasonable. For these aspects we refer to [Moser, 1996], where also investigations of theoretical properties of fuzzy partitions, such as redundancy, are provided.

For the case of a real interval $[a, b]$ one simple example is a fuzzy partition which consists of a chain of triangular fuzzy sets, where only two consecutive neighbors intersect with a maximal degree of 0.5. Such a fuzzy partition is uniquely determined by a (strictly) increasing sequence of $K = n - 1$ grid points $(a = x_0, x_1, \ldots, x_{n-1} = b)$ in the following way:

$$\begin{aligned}
\mu_{A_1}(x) &:= \begin{cases} \frac{x_1 - x}{x_1 - x_0} & \text{if } x \in [x_0, x_1] \\ 0 & \text{otherwise} \end{cases} \\
\mu_{A_i}(x) &:= \begin{cases} \frac{x - x_{i-1}}{x_{i-1} - x_{i-2}} & \text{if } x \in [x_{i-2}, x_{i-1}] \\ \frac{x_i - x}{x_i - x_{i-1}} & \text{if } x \in (x_{i-1}, x_i] \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i \in \overline{2, n-1} \\
\mu_{A_n}(x) &:= \begin{cases} \frac{x - x_{n-2}}{x_{n-1} - x_{n-2}} & \text{if } x \in [x_{n-2}, x_{n-1}] \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{4.7}$$

Figure 4.6 shows a typical example.

If we coded the values $x_0, \ldots, x_{n-1}$ directly, there would be a lot of invalid strings which represent non-increasing sequences. It can be shown, although it is intuitively clear, that the percentage of invalid strings grows with the number of values. An alternative way to encode such a partition is to encode the offsets $x_i - x_{i-1}$ (see also [Yubazaki *et al.*, 1995])

$$\boxed{c_{m,[\varepsilon,\delta]}(x_1 - x_0)} \cdots \boxed{c_{m,[\varepsilon,\delta]}(x_{n-2} - x_{n-3})},$$

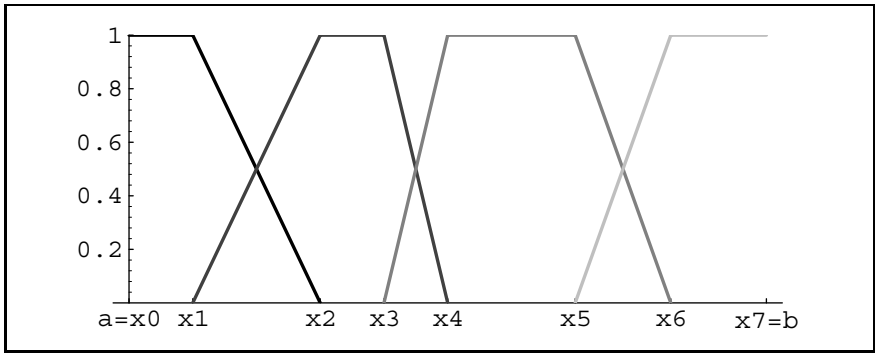


Figure 4.7: A fuzzy partition with $n = 4$ trapezoid parts

which yields a string of length $m \cdot (n-2)$. Again δ is a parameter greater than 0 which is chosen such that all the interesting cases are covered by the coding. One may argue that, after decoding, some values x_i can be greater than b. This problem can be overcome by enlarging the universe of discourse, concretely by enlarging b such that it is larger than $a + (n-1) \cdot \delta$ and by redefining

$$\mu_{A_n}(x) \quad := \quad \begin{cases} 0 & \text{if } x \in [x_0, x_{n-2}) \\ \frac{x - x_{n-2}}{x_{n-1} - x_{n-2}} & \text{if } x \in [x_{n-2}, x_{n-1}] \\ 1 & \text{otherwise,} \end{cases} \tag{4.8}$$

with additionally removing the restriction that $x_{n-1} = b$. Of course, this does not effect the input-output function of the fuzzy system.

The parameter ε is a lower bound for the offsets. If one also wants to model crisp transitions, ε has to be set to 0. Then we have to take special care of the degenerated case, where some offsets are 0 consecutively. Then one or more sets of the partition disappear at all, a problem whose solution depends on the needs of the concrete application.

This concept can also be generalized to trapezoid fuzzy sets. In this case the fuzzy partition is uniquely determined by a (strictly) increasing sequence

of $K = 2n$ points. The mathematical formulation is (compare with (4.7)):

$$\begin{aligned}
\mu_{A_1}(x) &:= \begin{cases} 1 & \text{if } x \in [x_0, x_1] \\ \frac{x_2 - x}{x_2 - x_1} & \text{if } x \in (x_1, x_2) \\ 0 & \text{otherwise} \end{cases} \\
\mu_{A_i}(x) &:= \begin{cases} \frac{x - x_{2i-3}}{x_{2i-2} - x_{2i-3}} & \text{if } x \in (x_{2i-3}, x_{2i-2}) \\ 1 & \text{if } x \in [x_{2i-2}, x_{2i-1}] \\ \frac{x_{2i} - x}{x_{2i} - x_{2i-1}} & \text{if } x \in (x_{2i}, x_{2i-1}) \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i \in \overline{2, n-1} \\
\mu_{A_n}(x) &:= \begin{cases} \frac{x - x_{2n-3}}{x_{2n-2} - x_{2n-3}} & \text{if } x \in (x_{2n-3}, x_{2n-2}) \\ 1 & \text{if } x \in [x_{2n-2}, x_{2n-1}] \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{4.9}$$

Figure 4.6 shows a typical example with $n = 4$. The method of coding the offsets as discussed above can be applied without any modifications.

4.1.3 Standard Fitness Functions

Although it is not possible to formulate a general recipe, which fits for all kinds of applications, there are some standard cases for which we can give standard fitness functions, which judge the performance of a fuzzy system. These cases have in common that they measure the discrepancy between actually obtained output with respect to a fixed configuration of the parameters and the desired output, under the assumption that it is known. We assume implicitely in the following that $f(\vec{v})$ is the fitness function which measures the performance of the fuzzy system depending on the parameter vector $\vec{v}$ which represents the configuration of the fuzzy sets. Without loss of generality, we restrict to fuzzy systems with one output. If there are more output variables, the fitness functions, which we discuss in the following, can be aggregated e.g. by addition.

Fuzzy Output on a Finite Set

Let $(p_1, \ldots, p_N)$ be a family of representative input values for which the desired output fuzzy sets $(O_1, \ldots, O_N)$ are known and let $O(\vec{v}, p)$ be the output fuzzy set which is obtained when the output of the fuzzy system is evaluated for input p with respect to the configuration $\vec{v}$. In the case that the output variable has a finite universe of discourse $\{x_0, \ldots, x_{n-1}\}$ (see also page 57), we can formulate the following fitness measure

$$f(\vec{v}) := \sum_{i=1}^{N} d(\vec{\mu}_{O_i}, \vec{\mu}_{O(\vec{v}, p_i)}), \tag{4.10}$$

with

$$\vec{\mu}_{(.)} := (\mu_{(.)}(x_0), \ldots, \mu_{(.)}(x_{n-1})),$$

where $d(.,.)$ is an arbitrary (pseudo)metric on $[0,1]^n$. This abstract definition may be a little confusing. The following special case, where the Euclidean norm $\|.\|_2$ is used as measure for the distance, illustrates better how this fitness can be computed:

$$f(\vec{v}) := \sum_{i=1}^{N} \left(\sum_{j=0}^{n-1} (\mu_{O_i}(x_j) - \mu_{O(\vec{v},p_i)}(x_j))^2 \right)^{\frac{1}{2}}. \tag{4.11}$$

Remark 4.4 A problem of this type is called "classification problem" (see also [Ishibuchi *et al.*, 1993]). The reason is simple: Such a fuzzy system maps values p to fuzzy subsets of a set of, for instance, linguistic labels $x_0, \ldots, x_{n-1}$. By the way, this is the kind our practical example (cf. 4.2) is of.

Fuzzy Output on an Interval

Analogously, let $(p_1, \ldots, p_N)$ be a family of representative input values for which the desired output fuzzy sets $(O_1, \ldots, O_N)$ are known $(O_i \in \mathcal{F}([a,b]))$. Let again $O(\vec{v}, p)$ be the output fuzzy set which is obtained when the output of the fuzzy system is evaluated for input p with respect to the configuration $\vec{v}$. Then a fitness function could be

$$f(\vec{v}) := \sum_{i=1}^{N} d(\mu_{O_i}, \mu_{O(\vec{v},p_i)}), \tag{4.12}$$

where $d(.,.)$ is a (pseudo)metric on $[a,b]^{[0,1]}$ or on a subset of $[a,b]^{[0,1]}$. In this case L_p-norms are commonly used. A typical example would be

$$f(\vec{v}) := \sum_{i=1}^{N} \left(\int_a^b (\mu_{O_i}(x) - \mu_{O(\vec{v},p_i)}(x))^2 \, dx \right)^{\frac{1}{2}}, \tag{4.13}$$

with the additional assumption that the membership functions of the O_i and $O(\vec{v}, p_i)$ are integrable for all i, which is not a serious restriction in practice.

Crisp Real-Valued Output

Another important problem, which occurs frequently in the optimization of fuzzy controllers, is the one of crisp real-valued output. For this case

let again $(p_1, \ldots, p_N)$ be a family of representative input values for which the desired output values $(o_1, \ldots, o_N)$ are known. If we denote the input-output functionof the controller with $\Phi_{\vec{v}}(x)$, an arbitrary (pseudo)metric on $[a, b]^N$, where $[a, b]$ is the output domain, can be used as fitness measure. A commonly used example is the Euclidean distance

$$f(\vec{v}) := \sqrt{\sum_{i=1}^{N} (o_i - \Phi_{\vec{v}}(p_i))^2}. \tag{4.14}$$

Another task, which differs a little bit from the ones we have discussed until now, is the approximation of given control surfaces by fuzzy controllers, where a complete description of the input-output surface is given and the parameters $\vec{v}$ should be found such that a certain distance in a function space is minimal. Although this is not only of theoretical, but also of practical interest, we refer to the literature here (e.g. [Moser, 1996]).

Remark 4.5 All the fitness measures, which have been introduced in this paragraph, are functions which are, of course, to be minimized. This is not so desirable if we want to apply genetic algorithms with proportional selection. However, it is rather easy to find an upper boundary or even the exact maximum for each one of these functions. If we denote this upper boundary with $f_{\max}$, then $\bar{f}(\vec{v}) := f_{\max} - f(\vec{v})$ is a function whose maximization is equivalent to the minimization of f. Depending on the concrete situation, it can also be of advantage to scale or to transform the fitness function.

4.1.4 Genetic Operations

The last important ingredients of genetic algorithms, which we must discuss before we can finally apply them to the optimization of fuzzy sets, are the genetic operations. For the case of single, independent fuzzy sets (see 4.1.1) various researchers have reported that the conventional methods, which we have already discussed in chapter 3, perform sufficiently well. An (incomplete) list of papers, which report that, would be [Karr, 1991], [Lee and Takagi, 1993a] and the other 1993 papers o M. A. Lee and H. Takagi, [Mitsubuchi *et al.*, 1993], [Surmann *et al.*, 1993], [Furuhashi *et al.*, 1995], [Lin and Chen, 1995], [Magdalena and Monasterio, 1995], [Shimojima *et al.*, 1995], and so on.

Crossing Over

For the case of fuzzy partitions the selection of an appropriate crossing over operator is a more subtle task. Figure 4.8 shows an example, where two fuzzy

partitions are crossed with one-point crossing over. Obviously, a change of one offset $x_i - x_{i-1}$ shifts all the values $x_i, \ldots, x_{K-1}$. From this point of view it might be clear that the existence or the development of building blocks (cf. 3.2 on page 52) is not so easy to guarantee, because schemata do not correspond to typicalities of the values x_i themselves but to typicalities of the offsets.

This problem can deteriorate the convergence behavior. However, experiments have shown (see also 4.2) that, if the fitness function is rather smooth and not very chaotic, some modified crossing over operations can speed up the convergence, but, on the contrary, if the fitness function is more chaotic, and this is the case where GAs are typically applied, such operations (one is demonstrated in example 4.6 below) speed up the convergence but not without badly increasing the risk of becoming trapped in a local maximum.

Example 4.6 One example of a crossing over technique specialized for fuzzy partitions could be the following:

1. decode the two parent strings such that the original sequences $X^1 = (x_0^1, \ldots, x_{K-1}^1)$ and $X^2 = (x_0^2, \ldots, x_{K-1}^2)$ are obtained,
2. choose a position l between 2 and $K-3$ randomly,
3. swap the tails $(x_{l+1}^1, \ldots, , x_{K-1}^1)$ and $(x_{l+1}^2, \ldots, , x_{K-1}^2)$ taking into account that the values must increase,
4. cross the values x_l^1 and x_l^2 in an appropriate way, e.g. $\tilde{x}_l^1 = \frac{2x_l^1 + x_l^2}{3}$ and $\tilde{x}_l^2 = \frac{x_l^1 + 2x_l^2}{3}$ again taking the restriction, that the values must increase, into account,
5. reencode the new sequences $\tilde{X}^1$ and $\tilde{X}^2$;

We can summarize that the crossing over operations must be chosen depending on the given problem. If nothing is known about the fitness function or if the fitness function is known to be chaotic with many local maxima, a standard crossing over operation should be used (see also [Yubazaki *et al.*, 1995]). If the fitness function is smooth and has not very many local maxima, a specialized crossing over operation should be used. If, in such a case, a conventional optimization method is applicable, a genetic algorithm should not be applied at all.

Mutation

Nearly everything, which has been said about the crossing over operations, also applies to the mutation operation. As apparent from figure 4.9, our

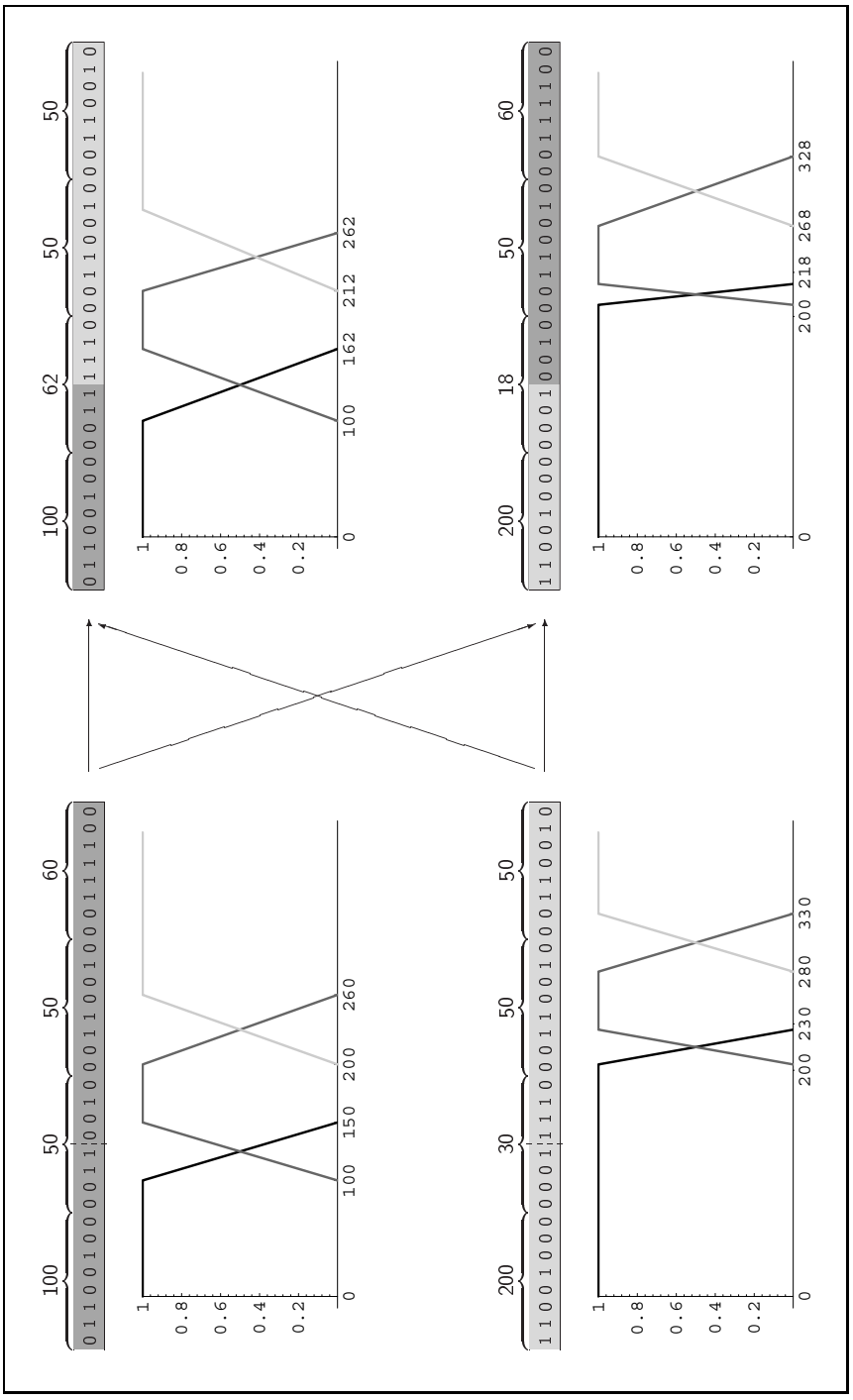


Figure 4.8: Example for one-point crossing over of fuzzy partitions ($n = 3$, $m = 8$)

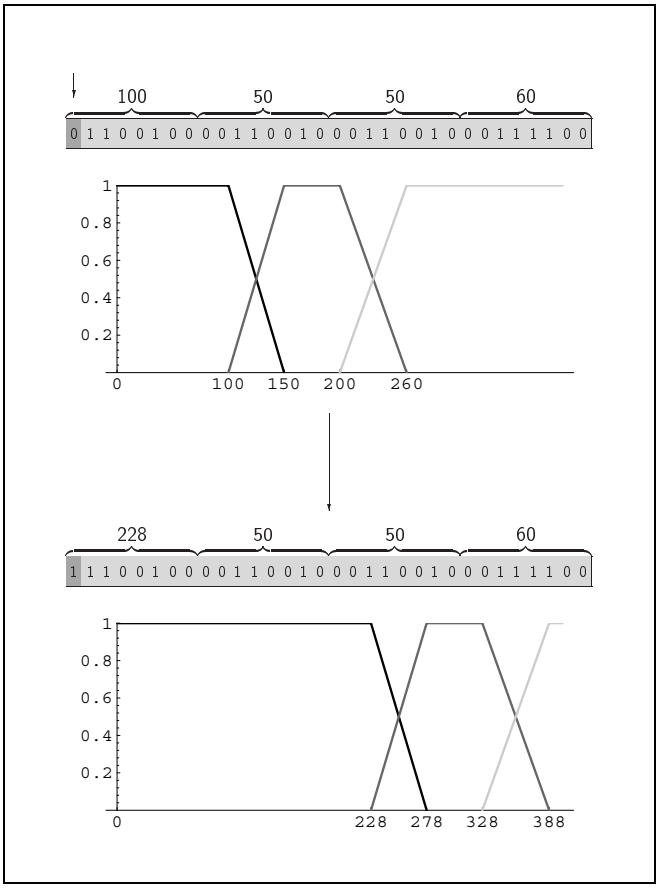


Figure 4.9: Mutating a fuzzy partition ($n = 3$, $m = 8$)

coding of fuzzy partitions has the property that a modification of a single bit can change the shape of the phenotype completely. So, in some sense, the coding does not preserve similarity relations — two similar genotypes can have completely different phenotypes. This seems to be bad at first glance. Surprisingly, experiments have demonstrated that exactly this property contributes a lot to the robustness of fuzzy GAs. It is intuitively clear that the risk of becoming trapped in a local maximum is not so critical, if the mutation operation can bring us far away from it.

4.1.5 Summary

The last paragraph has completed the list of things which are required for a genetic algorithm of type 3.2. If coding, fitness function, selection method, crossing over method, and mutation operator are chosen, a traditional ge-

netic algorithm, which operates on a fixed number of objects of a fixed size, can be applied (for a more general view of fuzzy GAs see [Buckley and Hayashi, 1994]).

In the next section a typical example of a fuzzy GA is introduced. It shows how a fuzzy GA works in practice and how its convergence can be improved.

4.2 An Application — Practical Results

This section gives an example which has, in fact, been part of an industrial project at the *FLLL*. The algorithm we will discuss within the following pages can be regarded as the most important result of the development of an inspection system for a silk-screen printing process. It computes a fuzzy segmentation of a given image into four different types of areas which are to be checked by applying different criteria. For us, the most interesting part is how the resulting fuzzy system is optimized with genetic algorithms.

4.2.1 Introduction

As anticipated above, we have to decide for each pixel of an image to which kind of area it belongs. Formally, this is a classification problem (see remark 4.4 on page 65, see also [Ishibuchi *et al.*, 1993] for reference).

The following four types were specified by experts of our partner company. For certain reasons, which can be explained with the special principles of the silk-screen printing process, it is sufficient to consider only these types:

Homogeneous area: uniformly colored area

Edge area: pixels within or close to visually significant edges

Raster: area which looks rather homogeneous from a certain distance, but which is actually obtained by printing small raster dots of two or even more colors

Aquarelle: rastered area with high chaotic deviations (e.g. small high-contrasted details in picture prints)

The magnifications in figure 4.10 show how theses areas typically look like. Of course, transitions between two or more of these areas are possible, hence a fuzzy model is recommendable.

First of all we should define more precisely what an image is:

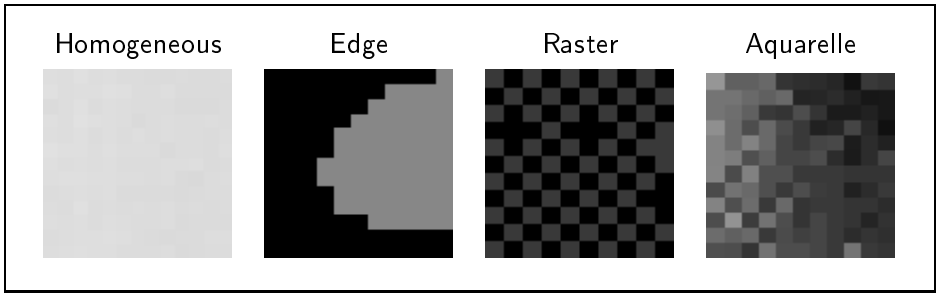


Figure 4.10: Magnifications of typical representatives of the four types

Definition 4.7 An $N \times M$ matrix of the form

$$((u_r(i,j), u_g(i,j), u_b(i,j)))_{i=1,\ldots,N}^{j=1,\ldots,M} \tag{4.15}$$

with 3-dimensional entries $(u_r(i,j), u_g(i,j), u_b(i,j)) \in \{0,\ldots,255\}^3$ is called a 24-bit color image of size $N \times M$. A coordinate pair (i,j) is called a pixel and the values $(u_r(i,j), u_g(i,j), u_b(i,j))$ are called the gray-values of pixel (i,j).

It is near at hand to use something like the variance or another measure for deviations to distinguish between areas which show only low deviations, such as homogeneous areas and rasters, and areas with high deviations, such as edge areas or aquarelles. On the contrary, it is intuitively clear that such a measure can never be used to separate edge areas from aquarelles, because any geometrical information is neglected. Experiments have shown that well-known standard edge detectors, such as the Laplacian or the Mexican Hat filter mask, cannot distinguish sufficiently if deviations are chaotic or anisotropic. Another possibility we also took into consideration was to use wavelet transforms (see [Daubechies, 1989] or [Stark, 1990]). Since the size of the image is approximately 1 Megabyte and the segmentation has to be done in at most three seconds, it is obvious that such highly advanced methods would require too much time. Finally, we found a fairly good alternative which is based on the discrepancy norm. This approach uses only, as filter masks like the Laplacian or the Mexican Hat also do, the closest neighborhood of a pixel. The following sketch shows how the neighbors of a fixed pixel (i,j) are enumerated. We can define the enumeration mapping l

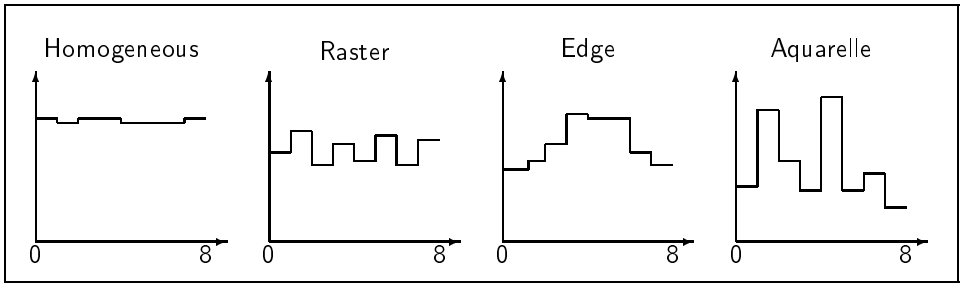
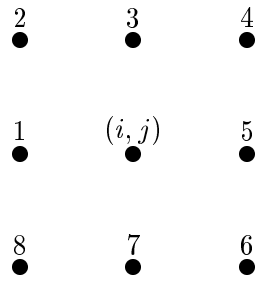


Figure 4.11: Typical gray-value curves of the form $(u_x(l(k))_{k\in\{1,\ldots,8\}}$

with the table besides:



k	$l(k)$
1	$(\quad i \quad, j-1\,)$
2	$(\,i-1\,, j-1\,)$
3	$(\,i-1\,, \quad j \quad)$
4	$(\,i-1\,, j+1\,)$
5	$(\quad i \quad, j+1\,)$
6	$(\,i+1\,, j+1\,)$
7	$(\,i+1\,, \quad j \quad)$
8	$(\,i+1\,, j+1\,)$

(4.16)

If we plot one color extraction with respect to this enumeration, concretely $u_x(l(k))_{k\in\{1,\ldots,8\}}$, where $x \in \{r, g, b\}$, we typically get curves like those ones shown in figure 4.11. From these sketches it can be seen easily that a measure for the deviations can be used to distinguish between homogeneous areas, rasters, and the other two types. On the contrary, the most eyecatching difference between aquarelles and edge areas is that edge areas show long connected peaks while aquarelles typically show chaotic, mostly narrow peaks. So, a method which judges the shape of the peaks should be used in order to separate edge areas from aquarelles. A simple but effective method for this purpose is the so-called discrepancy norm.

Definition 4.8

$$\begin{aligned} \|.\|_D : \quad \mathbb{R}^n &\longrightarrow \mathbb{R}^+ \\ (x_1, \ldots, x_n) &\mapsto \max_{1\le\alpha\le\beta\le n} \left| \sum_{i=\alpha}^{\beta} x_i \right| \end{aligned} \tag{4.17}$$

Lemma 4.9 $\|.\|_D$ is a norm.

Proof:

1. $\vec{x} = \vec{0} \Longleftrightarrow \|\vec{x}\|_D = 0$: trivial
2. $\|\lambda\vec{x}\|_D = |\lambda|\|\vec{x}\|_D$: trivial
3. $\|\vec{x} + \vec{y}\|_D \leq \|\vec{x}\|_D + \|\vec{y}\|_D$:

$$\begin{aligned}
\|\vec{x} + \vec{y}\|_D &= \max_{1\leq\alpha\leq\beta\leq n} \left|\sum_{i=\alpha}^{\beta}(x_i + y_i)\right| = \max_{1\leq\alpha\leq\beta\leq n} \left|\sum_{i=\alpha}^{\beta} x_i + \sum_{i=\alpha}^{\beta} y_i\right| \\
&\leq \max_{1\leq\alpha\leq\beta\leq n} \left(\left|\sum_{i=\alpha}^{\beta} x_i\right| + \left|\sum_{i=\alpha}^{\beta} y_i\right|\right) \\
&\leq \max_{1\leq\alpha\leq\beta\leq n} \left|\sum_{i=\alpha}^{\beta} x_i\right| + \max_{1\leq\alpha\leq\beta\leq n} \left|\sum_{i=\alpha}^{\beta} y_i\right| = \|\vec{x}\|_D + \|\vec{y}\|_D
\end{aligned}$$

■

In measure theory the discrepancy between two measures μ and ν on $\mathbb{R}$ is defined as $\mathcal{D}(\mu,\nu) := \max_{a\leq b} |\mu([a,b]) - \nu([a,b])|$. If we have two discrete measures $\overline{\mu}$ and $\overline{\nu}$ on the set $\{1,\ldots,n\}$, where $\overline{\mu}(i) =: x_i$ and $\overline{\nu}(i) =: y_i$, then $\mathcal{D}(\overline{\mu},\overline{\nu})$ and $\|\vec{x} - \vec{y}\|_D$ are equal (see [Weyl, 1916] or [Neunzert and Wetton, 1987]). Thus, we will call $\|.\|_D$ discrepancy norm on $\mathbb{R}^n$.

Obviously, the computation of $\|.\|_D$ by using the definition requires $\mathcal{O}(n^2)$ operations. The following theorem allows us to compute $\|.\|_D$ with linear speed.

Theorem 4.10

$$\|\vec{x}\|_D = \max_{1\leq\beta\leq n} X_\beta - \min_{1\leq\alpha\leq n} X_\alpha, \tag{4.18}$$

where the values

$$X_j := \sum_{i=1}^{j} x_i$$

denote the partial sums.

Proof: If we assign 0 to x_0 and x_{n+1} we can conclude that

$$\begin{aligned}
\|\vec{x}\|_D &= \max_{1\leq\alpha\leq\beta\leq n+1} \left|\sum_{i=\alpha}^{\beta} x_i\right| = \max_{1\leq\beta\leq n+1} \max_{1\leq\alpha\leq n+1} \left|\sum_{i=1}^{\beta} x_i - \sum_{i=1}^{\alpha-1} x_i\right| \\
&= \max_{1\leq\beta\leq n} \max_{1\leq\alpha\leq n} \left|\sum_{i=1}^{\beta} x_i - \sum_{i=1}^{\alpha} x_i\right| = \max_{1\leq\beta\leq n} \max_{1\leq\alpha\leq n} |X_\beta - X_\alpha| \\
&= \max_{1\leq\beta\leq n} X_\beta - \min_{1\leq\alpha\leq n} X_\alpha.
\end{aligned}$$

■

From (4.17) it can be seen easily that the more entries x_i with equal sign appear successively, the higher the value $\|\vec{x}\|_D$ is. This is exactly the reason

why the discrepancy norm can be used for our purpose of detecting visually significant edges.

4.2.2 The Fuzzy System

For each pixel (i, j) we consider the nearest eight neighbors enumerated as described above. Then we can use

$$\begin{aligned} v(i,j) := &\sum_{k=1}^{8} \left(u_r(l(k)) - \overline{r}\right)^2 + \sum_{k=1}^{8} \left(u_g(l(k)) - \overline{g}\right)^2 \\ &+ \sum_{k=1}^{8} \left(u_b(l(k)) - \overline{b}\right)^2 \end{aligned} \tag{4.19}$$

as a measure for the size of the deviations in the neighborhood of (i, j) and

$$\begin{aligned} e(i,j) :=& \| u_r(l(.)) - (\overline{r}, \ldots, \overline{r}) \|_D + \| u_g(l(.)) - (\overline{g}, \ldots, \overline{g}) \|_D \\ &+ \| u_b(l(.)) - (\overline{b}, \ldots, \overline{b}) \|_D \end{aligned} \tag{4.20}$$

as a measure whether the pixel is part of or lying adjacent to a visually significant edge, where $\overline{r}$, $\overline{g}$ and $\overline{b}$ denote the mean values

$$\overline{r} := \frac{1}{8} \sum_{k=1}^{8} u_r(l(k)),\ \overline{g} := \frac{1}{8} \sum_{k=1}^{8} u_g(l(k)),\ \overline{b} := \frac{1}{8} \sum_{k=1}^{8} u_b(l(k)).$$

Of course, e itself can be used as an edge detector. Figure 4.12 shows how good it works compared with the commonly used Mexican Hat filter mask.

The fuzzy decision is then done in a rather simple way: We have to compute the degrees of membership to which the pixel belongs to the four types of areas. Hence, the output of the fuzzy system is a vector

$$t(i,j) = (t_{\mathsf{H}}(i,j), t_{\mathsf{E}}(i,j), t_{\mathsf{R}}(i,j), t_{\mathsf{A}}(i,j)), \tag{4.21}$$

with $t_{\mathsf{H}}, t_{\mathsf{E}}, t_{\mathsf{R}}, t_{\mathsf{A}} \in [0, 1]$. Since the parameterization of the fuzzy systems is independent from the coordinates in our case, we just write v and e for the two inputs $v(i, j)$ and $e(i, j)$ which are treated as linguistic variables in the following. Experiments have shown that $[0, 600]$ and $[0, 200]$ are appropriate universes of discourse for v and e, respectively. We used simple fuzzy partitions for the fuzzy decomposition of the input space (see 4.1.2 on page 64). Their typical shape can be seen in figure 4.13.

Five rules, which cover all the possible cases, complete the fuzzy system:

IF	v is low			THEN	$t = \mathsf{H}$
IF	v is med	AND	e is high	THEN	$t = \mathsf{E}$
IF	v is high	AND	e is high	THEN	$t = \mathsf{E}$
IF	v is med	AND	e is low	THEN	$t = \mathsf{R}$
IF	v is high	AND	e is low	THEN	$t = \mathsf{A}$

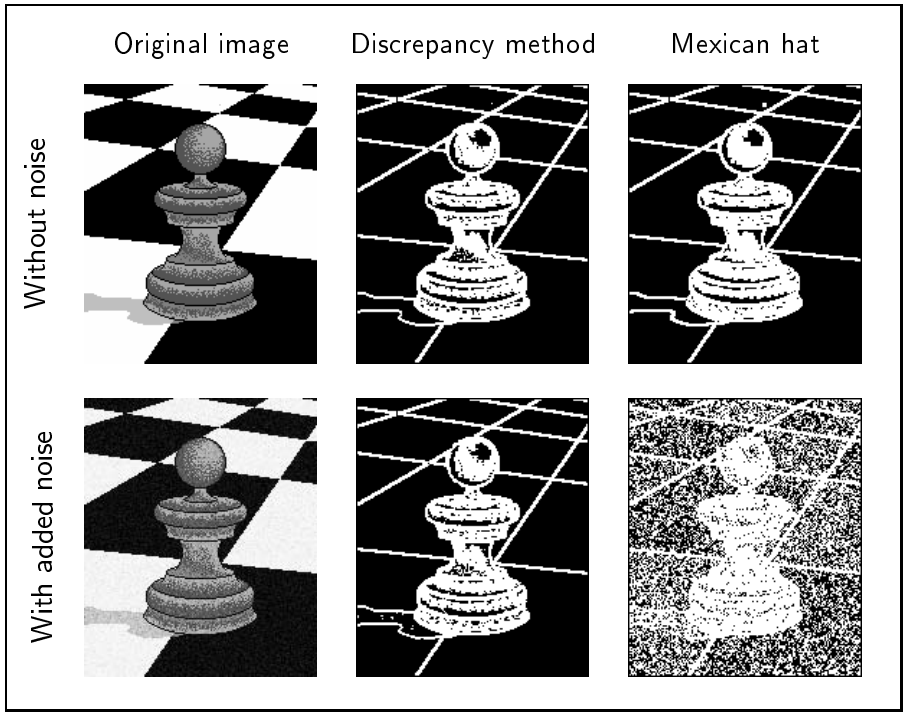


Figure 4.12: Comparison between e and a standard 3×3 filter mask

4.2.3 Optimization of the Fuzzy Partitions

As apparent from figure 4.13, the partitions depend on the six parameters v_1, v_2, v_3, v_4, e_1, and e_2. An interesting question is, of course, how to choose these values properly. In order to optimize them we need an objective criterion for judging the quality of the decision. Unfortunately, the specification of the four types is given in a verbal, imprecise form, which can hardly be formalized mathematically. However, it can be decided by a human whether the result of the segmentation algorithm for given parameters matches his/her own understanding of the four areas. So, we implemented a little painting program with pencils, rubbers, edge detection- and filling algorithms which can be used to prepare a segmentation by hand. This handmade segmentation can then be used as a reference.

Now assume that we have N sample pixels for which the pairs of input values $(\tilde{v}_k, \tilde{e}_k)_{k\in\{1,\ldots,N\}}$ are already computed and that we have a reference classification of these pixels $\tilde{t}(k) = (\tilde{t}_{\mathsf{H}}(k), \tilde{t}_{\mathsf{E}}(k), \tilde{t}_{\mathsf{R}}(k), \tilde{t}_{\mathsf{A}}(k))$, where $k \in \{1, \ldots, N\}$.[1] Then one possibility to define the performance (fitness) of the

[1]Since the geometry plays no role if the values $\tilde{v}$ and $\tilde{e}$ are already computed, we can switch to one dimensional indices here, what simplifies the formulas a little bit.

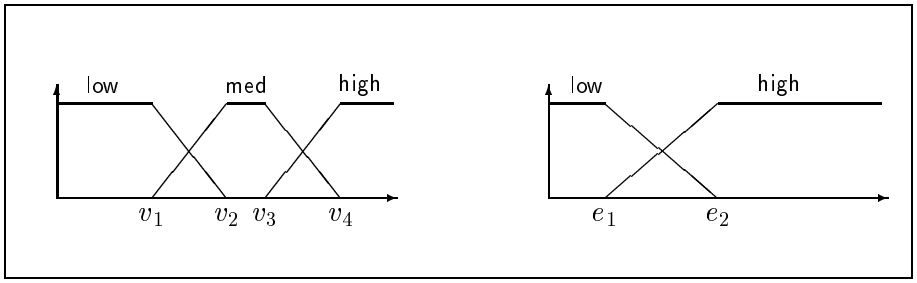


Figure 4.13: The linguistic variables v and e

fuzzy systems would be

$$\frac{1}{N}\sum_{k=1}^{N} d(t(k), \tilde{t}(k)), \tag{4.22}$$

where $t(k) = (t_{\mathsf{H}}(k), t_{\mathsf{E}}(k), t_{\mathsf{R}}(k), t_{\mathsf{A}}(k))$ are the classifications actually obtained by the fuzzy system for the input pairs $(\tilde{v}_k, \tilde{e}_k)$ with respect to the parameters v_1, v_2, v_3, v_4, e_1, and e_2; $d(.,.)$ is an arbitrary metric on $[0,1]^4$. The problem of this brute force approach is that the output of the fuzzy system has to be evaluated for each pair (v_k, e_k), even if many of these values are similar. In order to keep the amount of computation low, we "simplified" the procedure by a "clustering process" as follows: Choose a partition $(P_1, \ldots, P_K)$ of the input space and count the number $(n_1, \ldots, n_K)$ of sample points $\{p_1^j, \ldots, p_{n_j}^j\}$ each part contains. Then the desired classification of a certain part can be defined as

$$\tilde{t}_X(P_i) := \frac{1}{n_i}\sum_{j=1}^{n_i} \tilde{t}_X(p_j^i) \quad \text{with } X \in \{\mathsf{H}, \mathsf{E}, \mathsf{R}, \mathsf{A}\}. \tag{4.23}$$

If ϕ is a function, which maps each part to a representative value (e.g. its center of gravity), we can define the fitness as

$$f(v_1, \ldots, v_4, e_1, e_2) := \frac{50}{N}\sum_{i=1}^{K} n_i \cdot (2 - (*)), \tag{4.24}$$

with

$$(*) := \sum_{X \in \{\mathsf{H},\mathsf{E},\mathsf{R},\mathsf{A}\}} \left(\tilde{t}_X(P_i) - t_X(\phi(P_i))\right)^2.$$

If the number of parts is chosen moderately (e.g. a rectangular 64×32 net which yields $K = 2048$) the evaluation of the fitness function takes considerably less time than it would take if we used (4.22).

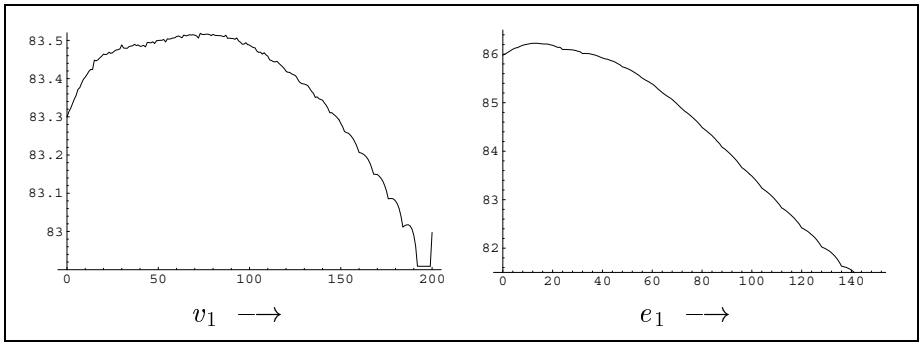


Figure 4.14: Cross sections of functions of type (4.24)

Remark 4.11 Note that in (4.24) the fitness is already transformed such that it can be regarded as a degree of matching between the desired and the actually obtained classification measured in percent. This value has then to be maximized.

Figure 4.14 shows cross sections of such a fitness function, where, in each case, five parameters are kept constant and one is varied. It can be seen easily that f is continuous but not necessarily differentiable and that there can be a lot of local maxima. Hence, it is not recommendable to use a conventional continous optimization method, such as gradient descent or a Newton-like method. Seemingly, the one and only way out of this trap was to use a probabilistic method. This requires, first of all, a coding of the parameters. We decided to use a coding, like the one proposed in 4.1.2 on page 64, which maps the parameters v_1, v_2, v_3, v_4, e_1 and e_2 to a string of six 8-bit integers $s_1, \ldots, s_6$ which range from 0 to 255. The following table shows how the encoding and decoding is done:

$$
\begin{array}{lcllcl}
s_1 & = & v_1 & v_1 & = & s_1 \\
s_2 & = & v_2 - v_1 & v_2 & = & s_1 + s_2 \\
s_3 & = & v_3 - v_2 & v_3 & = & s_1 + s_2 + s_3 \\
s_4 & = & v_4 - v_3 & v_4 & = & s_1 + s_2 + s_3 + s_4 \\
s_5 & = & e_1 & e_1 & = & s_5 \\
s_6 & = & e_2 - e_1 & e_2 & = & s_5 + s_6
\end{array}
$$

In order to compare the performance of various approaches, we considered the following methods:

Random Selection

Algorithm 4.12

choose a string G_1 *randomly*;
$f_1 := f(G_1)$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
    choose a string G_2 *randomly*;
    $f_2 := f(G_2)$;

    IF $f_2 > f_1$ **THEN**
    BEGIN
        $f_1 := f_2$;
        $G_1 := G_2$
    END
END

The results have shown that for a binary string length of 48 this is not a reasonable method at all.

Hill Climbing

In the strict sense, this method is a deterministic one. Since the initial point is chosen randomly, we also count it here.

Algorithm 4.13

choose a string G_1 *randomly*;
$f_1 := f(G_1)$;
$f_2 := \infty$;

WHILE $f_1 < f_2$ **DO**
BEGIN
    determine that neighbor string G_2 *of* G_1 *with highest fitness* f_2;

    IF $f_2 > f_1$ **THEN**
    BEGIN
        $f_1 := f_2$;
        $G_1 := G_2$
    END
END

Simulated Annealing

The simulated annealing algorithm is a powerful probabilistic optimization technique which has been widely used during the last years. It imitates the solidification of crystals under slowly decreasing temperature. The main idea is that every atom tends to reach a state of thermic equilibrium. Thermic equilibrium on a certain temperature level is reached if the temperature of a randomly chosen neighbor atom is Boltzmann distributed (for more details see [van Laarhoven and Aarts, 1987] or [Otten and van Ginneken, 1989]). This approach goes back to N. Metropolis who proposed an algorithm for the efficient simulation of the evolution of a solid-to-thermal equilibrium (cf. [Metropolis *et al.*, 1953]) and was later discovered to be an appropriate method for solving combinatorial optimization problems (early publications were [Kirkpatrick *et al.*, 1983] and [Černy, 1985]).

Algorithm 4.14

$k := 0$;
set initial temperature T;
choose a G_1 *randomly*;
$f_1 := f(G_1)$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
  REPEAT
    choose neighbor string G_2 *randomly*;
    $f_2 := f(G_2)$;

    IF $f_2 > f_1$ **THEN**
    BEGIN
      $G_1 := G_2$;
      $f_1 := f_2$
    END
    ELSE IF $\mathsf{Random}[0,1] < e^{\frac{f_2 - f_1}{T}}$ **THEN**
    BEGIN
      $G_1 := G_2$;
      $f_1 := f_2$
    END

  UNTIL *state of equilibrium is reached sufficiently closely ;*

  $T := \phi(T, k)$;
  $k := k + 1$
END

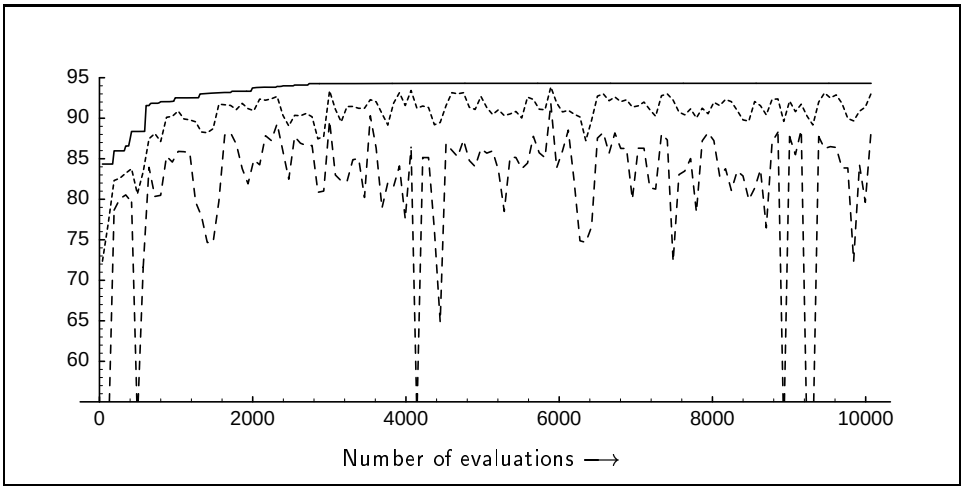


Figure 4.15: The performance graph of a fuzzy GA

The Raw GA

First of all, we took a GA as proposed in algorithm 3.6 with the coding presented above, a population size of 20, one-point crossing over with probability 0.15, and standard mutation, where each bit is modified with equal probability p_{M} (0.005 in this case). Figure 4.15 shows a typical performance graph of a genetic algorithm of this kind.

Hybrid Genetic Algorithm

Genetic algorithms are methods which are more or less "blind" in their search for the optimal solution. So, it may be of interest what happens if a GA is additionally supported by another method. We tried out some combinations of the raw GA and the hill climbing method.

Results

All these algorithms are probabilistic methods, therefore, their results are not well-determined, they can differ randomly within certain boundaries. So, we tried out each one of them 20 times for one certain problem in order to obtain more information about their average behavior. For the given problem we found out that the maximal degree of matching between the reference classification and the classification actually obtained by the fuzzy system was 94.3776% . The table in figure 4.16 shows the results in detail.

	$f_{\mathbf{max}}$	$f_{\mathbf{min}}$	$\bar{f}$	σ_f	It
Hill Climbing	94.3659	89.6629	93.5536	1.106	862
Simulated Annealing	94.3648	89.6625	93.5639	1.390	1510
Improved Simulated Annealing	94.3773	93.7056	94.2697	0.229	21968
GA	94.3760	93.5927	94.2485	0.218	9910
Hybrid GA (elite)	94.3760	93.6299	94.2775	0.207	7460
Hybrid GA (random)	94.3776	94.3362	94.3693	0.009	18631

Figure 4.16: Some results

The hill climbing method with a random selection of the initial string converged rather quickly. Unfortunately, it was always trapped in a local maximum, but never reached the global solution (at least in these 20 trials).

The simulated annealing algorithm showed similar behavior at the very beginning, when we used $\phi(T,k) := T \cdot 0.9962$. T was set to 2 initially. After some experiments with the parameters the performance could be improved remarkably (with $\phi(T,k) := T \cdot 0.9914$ but more iterations in the inner loop).

The raw genetic algorithm looked pretty good from the beginning, but it seemed inferior to the improved simulated annealing.

Next, we tried a hybrid GA, where we kept the genetic operations and parameters of the raw GA, but every 50-th generation the best-fitted individual was taken as initial string for a hill climbing method. Although the performance increased, the hybrid method still seemed to be worse than the improved simulated annealing algorithm. The reason that the effects of the modification were not so dramatical might be that the probability is rather high that the best individual is already a local maximum. So we modified the procedure again. This time a *randomly chosen individual* of every 25-th generation was used as initial string of the hill climbing method. The result exceeded the expectations by far. The algorithm was, in all cases, nearer to the global solution than the improved simulated annealing was (compare with the table in figure 4.16), but, surprisingly, sufficed with less invocations of the fitness function.

Figure 4.17 shows a graphical representation of the results. Each curve

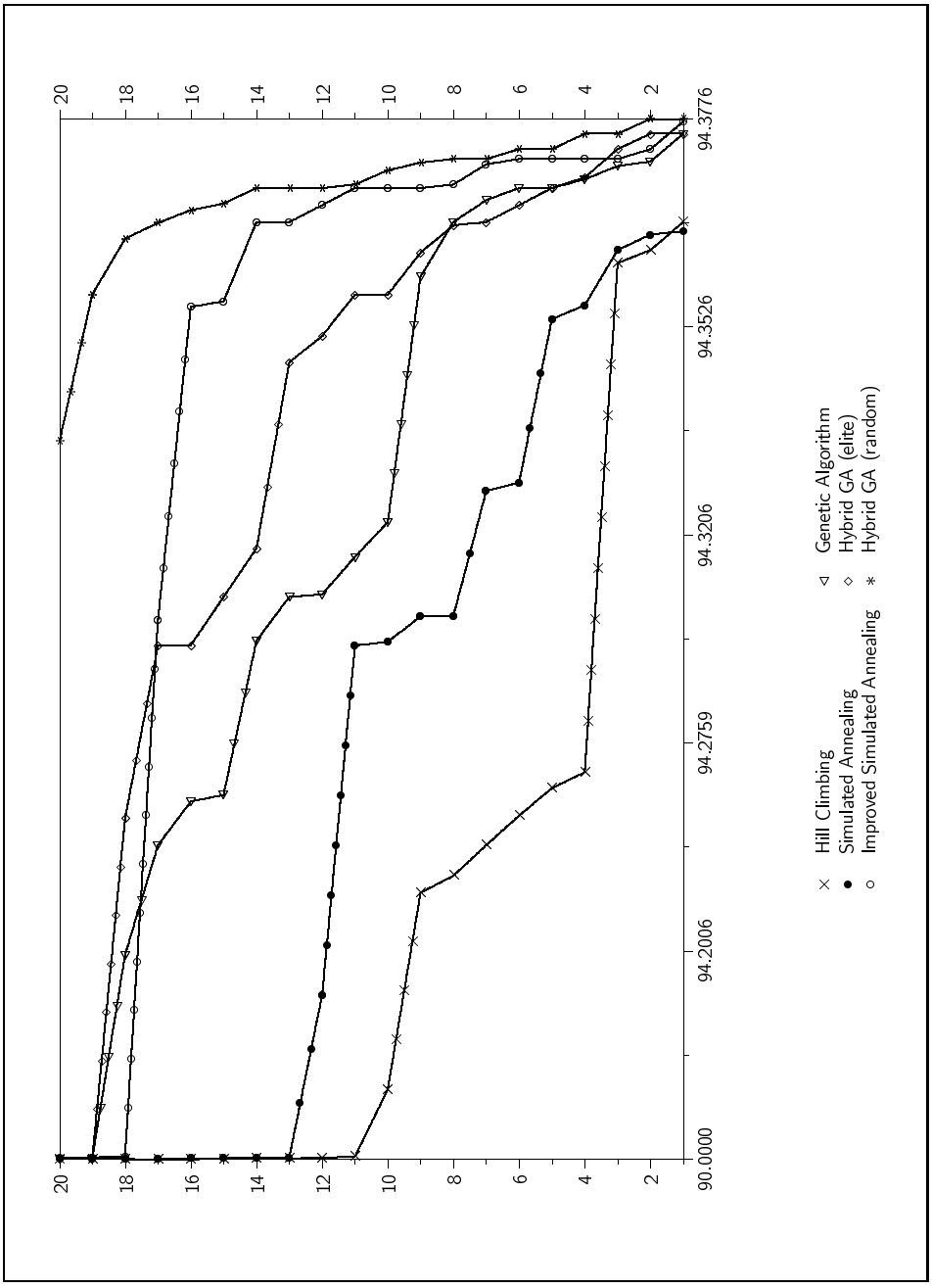


Figure 4.17: A graphical representation of the results

in this graph corresponds to one algorithm. Such a curve shows, for a given fitness value x, how many of the 20 solutions had a fitness higher or equal to x. It can be seen easily from this graph that the hybrid GA with random selection brought the best results. The x axis is not a linear scale in this figure. It was transformed in order to make small differences visible.

4.3 Conclusion

From the examples above, we have seen that genetic algorithms are costly but effective. The reason for the robustness and the stable convergence behavior is that GAs always have a wider perspective of the problem, because they keep a whole population of points in mind instead of only single points, as the conventional methods do.

We have seen further that intelligent combinations of conventional methods and genetic algorithms can yield significant improvements in terms of speed and the quality of the solutions.

Chapter 5

Acquiring Rulebases with Genetic Algorithms

There are two concepts within fuzzy logic which play a central role in its applications. The first is that of a linguistic variable, that is, a variable whose values are words or sentences in a natural or synthetic language. The other is that of a fuzzy if-then rule in which the antecedent and consequent are propositions containing linguistic variables. The essential function served by linguistic variables is that of granulation of variables and their dependencies. In effect, the use of linguistic variables and fuzzy if-then rules results — through granulation — in soft data compression which exploits the tolerance for imprecision and uncertainty. In this respect, fuzzy logic mimics the crucial ability of the human mind to summarize data and focus on decision-relevant information.

Lotfi Asker Zadeh in [Zadeh, 1993]

This quotation by L. A. Zadeh expresses brilliantly what the core of fuzzy logic is. As we have already seen (more often than once), he recalls that the power of fuzzy logic lies in the separation of rules and their concrete meaning (see also page 56). In the previous chapter we have dealt with the optimization of the latter component. Now we give an introduction to methods for the acquisation of rulebases with genetic algorithms. We assume that the linguistic variables are completely specified. The simultaneous optimization of rules and their meaning will be discussed in 6.1.

While we have only discussed methods for offline optimization until now, the optimization of rulebases can be done in two ways. Besides techniques,

which are applied offline, methods for learning rules continuously have come into fashion. One prominent approach is the fuzzification of a Holland classifier system which bases upon the application of genetic operations to generate rules. Such methods will be discussed in 5.2. Before that we take a look at offline techniques.

5.1 Offline Optimization of Rulebases

First of all, let us recall the definition of a fuzzy GA (see page 57): A fuzzy genetic algorithm is a genetic algorithm which is applied to the optimization of some parameters of a fuzzy system. Of course, this definition also includes the offline optimization of rulebases. However, other names have become commonly used: Offline optimization of rulebases is often nicknamed the "Pitt Approach" after Pittsburgh, the town where the first decision system, whose parameters were tuned with genetic algorithms, was written (S. F. Smith's poker player, see [Goldberg, 1989]). According to that, a fuzzy system, which employs genetic offline learning, is often called a fuzzy classifier system of the Pittsburgh type. Generally, *a classifier system is a machine learning system which learns rules in order to guide its own performance in an arbitrary environment* (see [Holland, 1986] or [Geyer-Schulz, 1995]). Classifiers are nothing else but ordinary if-then rules. The name classifier comes from the capability of rules to classify inputs into message sets (compare with the sets R_j in (2.21) on page 13).

5.1.1 Fixed-Length Representations

If we find a method for encoding rulebases into a string of a fixed length, all the genetic methods we have previously dealt with are applicable with only little modifications. Of course, we have to assume in this case that the numbers of verbal values of the linguistic variables, which are involved, are finite. For simplicity, without much loss of generality, we only deal with the case that the set of verbal values just contains adjectives.

The simpliest case is that of coding a complete rulebase, which covers all the possible cases, into a matrix (a tensor in the case of more than two input variables). For this purpose, consider a rulebase of the following form (the generalization to more than two input variable is straightforward):

$$\mathsf{IF} \quad x_1 \text{ is } A_i \text{ AND } x_2 \text{ is } B_j \quad \mathsf{THEN} \quad y \text{ is } \tilde{C}_{ij} \tag{5.1}$$

A_i and B_j are verbal values of the variables x_1 and x_2, respectively. All the values A_i are pairwise different, analogously for the values B_j; i ranges from 1 to N_1, the total number of verbal values of variable x_1; j ranges from 1

to N_2, the total number of verbal values of variable x_2. The values $\tilde{C}_{ij}$ are arbitrary elements of the set of pairwise different verbal values $\{C_1, \ldots, C_{N_y}\}$ of linguistic variable y. Obviously, such a rulebase is uniquely represented by a matrix, a decision table, where the position of a consequent value determines to which premise it belongs,

	B_1	$\cdots$	B_{N_2}
A_1	$\tilde{C}_{11}$	$\cdots$	$\tilde{C}_{1N_2}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
A_{N_1}	$\tilde{C}_{N_1 1}$	$\cdots$	$\tilde{C}_{N_1 N_2}$.

If we associate that number k_{ij} between 1 and N_y, for which $\tilde{C}_{ij}$ equals $C_{k_{ij}}$, with each $\tilde{C}_{ij}$, we can easily encode such a rulebase by putting all the binary representations of the values $k_{ij} - 1$ into a string of length $N_1 \cdot N_2 \cdot K$ with $K := \log_2 N_y$.

For the method above, genetic algorithms as presented in algorithms 3.2 or 3.6 are suitable. We only have to take care that mutation and crossing over can yield values higher than $N_y - 1$, if N_y is not a power of 2. Of course, the fitness functions, which we have introduced in 4.1.3, can also be used without any modifications. Such a method is for example applied in [Thrift, 1991]. A related approach is that of Lee and Takagi which we will disucss in 6.1.1.

It is easy to see that the approach above works consequent-oriented, what means, that the premises are fixed, and, therefore, the consequent values must be acquired. Such an idea can only be applied to optimization of complete rulebases which are, in more complex applications, not so easy to interpret. Moreover, complete rulebases can have the disadvantages that they require a lot of storage for, in many cases, rules which are not necessary. On the contrary, in many applications, especially in control applications, it is enough to have an incomplete rulebase which consists of a certain number of rules which cover the input space sufficiently well.

The acquisation of incomplete rulebases is a task, which is not so easy to solve with representations of fixed length. We will come to that later in 5.1.2 and 5.2. Nevertheless, there is at least one method, which can be applied with fixed-length representations. It can be found in [Gonzalez *et al.*, 1993], where a genetic algorithm is applied to the problem of learning structures of single rules.

Typically, incomplete rulebases consist of a comparatively small number of rules of a more general shape, rules which do not necessarily specify restrictions for all input variables (as in (5.1)), so-called generalizing rules, rules which do not only employ AND as connective, but also OR, or unary operators, such as NOT. It is intuitively clear that such a variety of expressions cannot be represented sufficiently well by binary strings of a fixed

length. However, in [Gonzalez *et al.*, 1993] a subset of such rules, which might be sufficient for many applications, is considered.

For this purpose we consider an atomic fuzzy system with n input variables $x_1, \ldots, x_n$ and one output variable y. For each input variable x_i we have a finite set D_i of verbal values $D_i := \{A_{i1}, \ldots, A_{ik_i}\}$ with which we associate fuzzy sets. Analogously, we have a finite set of verbal labels $B_1, \ldots, B_{k_y}$ for output variable y. Then we consider a rulebase of the form

$$\begin{array}{llll}
\mathsf{IF} & \mathrm{Ant}_{11} & \mathsf{THEN} & y \text{ is } B_1 \\
\vdots & \vdots & \vdots & \vdots \\
\mathsf{IF} & \mathrm{Ant}_{1m_1} & \mathsf{THEN} & y \text{ is } B_1 \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\mathsf{IF} & \mathrm{Ant}_{n_y 1} & \mathsf{THEN} & y \text{ is } B_{n_y} \\
\vdots & \vdots & \vdots & \vdots \\
\mathsf{IF} & \mathrm{Ant}_{n_y m_{n_y}} & \mathsf{THEN} & y \text{ is } B_{n_y}.
\end{array} \tag{5.2}$$

Apparently, we have fixed a number of rules $\sum_{i=1}^{n_y} m_i$ in advance, where m_i premises (antecedents) Ant_{ij} yield the consequent B_i.

In this approach, antecedents of the shape

$$x_1 \text{ is } \tilde{A}_1 \text{ AND } \cdots \text{ AND } x_n \text{ is } \tilde{A}_n, \tag{5.3}$$

are considered, where $\tilde{A}_i$ are lists of labels contained in D_i. The semantic interpretation of such a list is the fuzzy union of the fuzzy sets which are associated with the labels in the list. Concretely, if $\tilde{A}_i = \{A_{ij_1}, \ldots, A_{ij_K}\}$, the expression x_i is $\tilde{A}_i$ can be read as

$$x_i \text{ is } A_{ij_1} \text{ OR } \cdots \text{ OR } x_i \text{ is } A_{ij_K}.$$

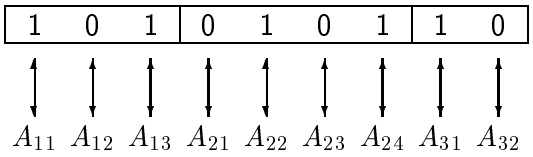
For encoding a rulebase of type (5.2), we must, first of all, find a coding for the lists $\tilde{A}_i$. This can easily be done by associating a binary string of length n_i with each $\tilde{A}_i$. Position j is then 1, if A_j is contained in $\tilde{A}_i$, 0 otherwise. If the string, which represents $\tilde{A}_i$, only contains zero entries, this part of the premise is not evaluated. Hence, it is also possible to represent generalizing rules.

An antecedent as proposed in (5.3) can then be encoded by concatenating the strings which represent the lists $\tilde{A}_i$. Finally, the rulebase (5.2) is encoded by concatenating the codings of all the antecedents Ant_{ij}.

Example 5.1 Suppose we have three inputs x_1, x_1, and x_3 with $D_1 := \{A_{11}, A_{12}, A_{13}\}$, $D_2 := \{A_{21}, A_{22}, A_{23}, A_{24}\}$, and $D_3 := \{A_{31}, A_{32}\}$. Then the antecedent

$$x_1 \text{ is } \{A_{11}, A_{13}\} \text{ AND } x_2 \text{ is } \{A_{22}, A_{24}\} \text{ AND } x_3 \text{ is } \{A_{31}\}$$

is encoded as



Obviously, all the conventional genetic operations, including selection, crossing over, mutation, are applicable. Although the standard fitness functions can also be used, it can be of advantage to additionally incorporate terms, which judge the simplicity of the rulebase, into the fitness function.

This approach has, compared with the representation of complete rulebases, the advantage that it allows generalizing rules which contribute much to the compactness and interpretability of rulebases. Nevertheless, we can state the following disadvantages:

- The number of rules must be fixed in advance.
- The application of the connective OR is limited to the union of sets which belong to the same input variable.
- More advanced constructs, such as unary operators like NOT or adverbs cannot be allowed.

The next section is going to deal with a paradigm which overcomes these disadvantages.

5.1.2 Fuzzy Genetic Programming

Not surprisingly, all kinds of fuzzy GAs, which incorporate genetic programming techniques, are subsumed under the term "Fuzzy Genetic Programming" (fuzzy GP). This field is a comparatively new one. Most of the theory goes back to A. Geyer-Schulz (see [Geyer-Schulz, 1995]) who also implemented the first application, where he tried to improve stock management strategies with fuzzy GP ([Geyer-Schulz, 1996]).

Within this promising approach, all kinds of constructs for representing fuzzy knowledge, such as adverbs, connectives, and so on, can be used. The first thing we need is a rule language. Typically, a rule language for a fuzzy system with n inputs $x_1,\ldots,x_n$ in Backus-Naur form looks as follows:

$$
\begin{array}{rcl}
\text{<rule>} & := & \text{“IF” <premise> “THEN” <conclusion>;} \\
\text{<premise>} & := & \text{<conditional>} \mid \\
 & & \text{“(” <unary> <premise> “)” } \mid \\
 & & \text{“(” <premise> <connective> <premise> “)” ;} \\
\text{<conditional>} & := & \text{“}x_1\text{” “is” <expr}_1\text{>} \mid \cdots \mid \text{“}x_n\text{” “is” <expr}_n\text{> ;} \\
\text{<unary>} & := & \text{“NOT” ;} \\
\text{<connective>} & := & \text{“AND” } \mid \text{ “OR” ;} \\
\text{<conclusion>} & := & \text{“}y\text{” “is” <expr}_y\text{> ;}
\end{array}
$$

where the expressions $<\text{expr}_1> \ldots <\text{expr}_n>$ and $<\text{expr}_y>$ are verbal values of the linguistic variables $x_1 \ldots x_n$ and y, respectively. The syntax of these expressions is specified by the grammars which produce the verbal values of the linguistic variables (compare with definition 2.15 and page 22). Since these grammars are also given in BNF, they can easily be implanted in the grammar above. It is easy to see that the grammar above provides a higher generality than all the approaches we have considered previously.

Before we can apply the universal genetic programming techniques, which we have introduced in 5.1.2, we have to find a representation for a whole rulebase. Of course it is again possible to fix a certain number m of rules in advance. Then an appropriate syntax for a rulebase would be

$$
\text{<rulebase>} \quad := \quad \text{“(” } \underbrace{\text{<rule> “,” } \ldots \text{ “,” <rule>}}_{m \text{ times}} \text{ “)” ;}
$$

The more general way to represent a rulebase, which does not suffer from the problem that the size must be fixed in advance, is to allow an arbitrary number of rules in the rulebase:

$$
\begin{array}{rcl}
\text{<rulebase>} & := & \text{“(” <rulelist> “)” ;} \\
\text{<rulelist>} & := & \text{<rule>} \mid \text{“,” <rulelist> ;}
\end{array}
$$

Furthermore, if the set of verbal values of the linguistic variable y is a finite set of adjectives, it is, under some additional assumptions, possible to suffice with a finite number of rules.

Lemma 5.2 *Consider an atomic fuzzy system with the notations as above. Suppose that its output is computed with the Mamdani inference, where* $T_1 = T_2 = T_{\mathrm{M}}$ *and* $S_1 = S_2 = S_{\mathrm{M}}$*. If the set of verbal values of the output variable* y *is a finite set of* n_y *labels* $D_y := \{B_1, \ldots, B_{n_y}\}$*, an arbitrary rulebase with a finite number* m *of rules of the form*

$$
\mathsf{IF} \quad \mathit{Expr}_j \quad \mathsf{THEN} \quad y \text{ is } \tilde{B}_j, \tag{5.4}
$$

where $\tilde{B}_j \in D_y$*, can be rewritten as a rulebase with at most* n_y *rules with pairwise different consequent values without changing the input-output function.*

Proof: With the setting

$$D'_y := \{A \in D_j | \exists j \in \overline{1,m}\ A = \tilde{B}_j\} \subseteq D_y, \quad n'_y := |D'_y| \leq |D_y| = n_y$$

and if we denote the elements of D'_y with $B'_1,\ldots,B'_{n'_y}$, the rulebase (5.4) can be rewritten as

$$\mathsf{IF} \quad \mathit{Expr}'_k \quad \mathsf{THEN} \quad y \text{ is } B'_k \qquad k \in \overline{1,n'_y}, \tag{5.5}$$

where Expr'_k is the fuzzy disjunction of all expressions Expr_j which have B'_k as consequent value. More precisely

$$\mathit{Expr}'_k = \mathit{Expr}_{i_{k1}} \ \mathsf{OR} \ \cdots \ \mathsf{OR} \ \mathit{Expr}_{i_{kl_k}},$$

where $\{i_1,\ldots,i_k\} \equiv \{j|\tilde{B}_j = B'_k\}$ and $l_k := |\{i_1,\ldots,i_k\}|$. Obviously, the rulebase above has at most n_y rules with pairwise different consequent values.

Now, let $\vec{x}$ be an arbitrary input vector and let z be an arbitrary element of Y, the universe of discourse of the output variable y. Then the output fuzzy set of rulebase (5.4) with respect to input $\vec{x}$ is

$$\mu_{C_1(\vec{x})}(z) = S_{\mathbf{M}}\big(T_{\mathbf{M}}(\mu_{\tilde{B}_1}(z), \mathit{Expr}_1(\vec{x})), \ldots T_{\mathbf{M}}(\mu_{\tilde{B}_m}(z), \mathit{Expr}_m(\vec{x}))\big).$$

The output fuzzy set of rulebase (5.5) is given as

$$\begin{aligned}\mu_{C_2(\vec{x})}(z) &= S_{\mathbf{M}}\big(T_{\mathbf{M}}(\mu_{B'_1}(z), \mathit{Expr}'_1(\vec{x})), \ldots T_{\mathbf{M}}(\mu_{B'_{n'_y}}(z), \mathit{Expr}'_{n'_y}(\vec{x}))\big)\\ &= S_{\mathbf{M}}\big(\ T_{\mathbf{M}}(\mu_{B'_1}(z), S_{\mathbf{M}}(\mathit{Expr}_{i_{11}}(\vec{x}),\ldots,\mathit{Expr}_{i_{1l_1}}(\vec{x}))),\ldots\\ &\qquad T_{\mathbf{M}}(\mu_{B'_{n'_y}}(z), S_{\mathbf{M}}(\mathit{Expr}_{i_{n'_y1}}(\vec{x}),\ldots,\mathit{Expr}_{i_{n'_yl_{n'_y}}}(\vec{x})))\big)\end{aligned}$$

Together with

$$\min(x,\max(y,z)) = \max(\min(x,y),\min(x,z))$$

and

$$\max(\max(x_1,x_2),\max(x_3,x_4)) = \max(x_1,x_2,x_3,x_4),$$

the assertion follows. ■

Example 5.3 In order to illustrate the transformation process described in the proof of lemma 5.2, consider the following rulebase with two inputs ($D_1 = D_2 = \{$"neg", "zero", "pos"$\}$ and $D_y = \{$"low", "medium", "high"$\}$):

IF	x_1 is "neg"	THEN	y is "low"
IF	x_1 is "zero" AND x_2 is "pos"	THEN	y is "low"
IF	x_1 is "pos"	THEN	y is "medium"
IF	x_1 is "zero" AND x_2 is "neg"	THEN	y is "medium"
IF	x_1 is "zero" AND x_2 is "zero"	THEN	y is "high"

The transformed variant is then

IF	x_1 is "neg" OR (x_1 is "zero" AND x_2 is "pos")	THEN	y is "low"
IF	x_1 is "pos" OR (x_1 is "zero" AND x_2 is "neg")	THEN	y is "medium"
IF	x_1 is "zero" AND x_2 is "zero"	THEN	y is "high".

If the assumptions of lemma 5.2 are fulfilled, we can suffice with a representation with a fixed number of rules, n_y concretely. In this case, as already shown in the previous section, we have to find the antecedent expressions for a fixed set of consequences.

To all the representations, which we have shown here, the genetic operations can be applied as usual. It can be of advantage to incorporate mechanisms into the fitness function which additionally take the complexity and the number of the rules into account. Moreover, it is useful to simplify the expressions after each generation in order to avoid wild growth of the derivation trees. If simplification is desired, the operations (t-norms and t-conorms) should be chosen such that some derivation laws, such as the De-Morgan law, are fulfilled.

More, especially theoretical details on fuzzy genetic programming can be found in [Geyer-Schulz, 1995], where also a global convergence proof is provided.

5.2 Online Learning of Fuzzy Rules

The ideas in the previous section have in common that (1) the genetic algorithms operate on whole rulebases, they work with populations of rulebases, and (2) rulebases are judged globally, i.e. the performance of whole rulebases is evaluated by the fitness function. If the systems are judged globally, no complicated examination which rules are responsible for success or failure, what requires profound knowledge about the environment, has to be done. This seems to be an advantage at first glance, but, in fact, the convergence of such methods can be weak, because single obstructive rules can deteriorate the fitness of the whole rulebase, which could contain very useful, well-performing rules. Furthermore, we have mentioned that genetic algorithms are capable of finding fairly good solutions, but local refinement can take a lot of time. Another aspect is that it can be difficult to define a global quality measure which provides enough information to guide a genetic algorithm to the solution.

Therefore, it is, of course, of interest to consider methods which observe the behavior of the system throughout a certain period of time adjusting the rules according to local payoff from the system. Consider for example the game of chess in order to demonstrate the difference between the two

approaches. A global quality measure could be the percentage of successes in a large number of games or, more specifically, the number of moves it took to be successful in the case of success and the number of moves it had been possible to postpone the winning of the opponent in the case of failure. It is easy to see that such information provides only a scarce foundation for learning chess, even if more detailed information, such as the number of pieces captured, is involved. On the contrary, it is easier to learn the principles of chess, when the direct effect of the application of a certain rule can be observed immediately. The problem, not only in the case of chess, is that early moves can also contribute much to a final success. We will come to that later.

In general, machine learning systems, which employ a genetic algorithm for the manipulation of single rules, in a way as described above are called classifier systems of the Michigan type. Figure 5.1 shows the typical architecture of such a system. The main components are:

1. A production system containing a rulebase which processes incoming messages from the environment and sends output messages to the environment

2. An apportionment of credit system which receives payoff from the environment and determines which rules had been responsible for that feedback; this component assigns strength values to the single rules in the rulebase. These values represent the performance and usefulness of the rules.

3. A genetic algorithm which combines well-performing rules to new ones with respect to their strength values.

Obviously, the learning task is divided into two subtasks — the judgment of already existing and the discovery of new rules.

Before we turn to classifier systems, which actually learn fuzzy rules in such an online process, just to sharpen our understanding, we discuss a simple crisp variant which has been examined very well — the so-called Holland classifier system.

5.2.1 The Holland Classifier System

A Holland classifier system is a classifier system of the Michigan type which processes binary messages of a fixed length through a rulebase whose rules are adapted according to the response of the environment. There are a lot of different notations in the literature (e.g. [Holland, 1986], [Holland, 1992], [Holland *et al.*, 1986], [Holland *et al.*, 1987], or [Geyer-Schulz, 1995]), we have developed our own view which, in some sense, merges the differently occurring approaches.

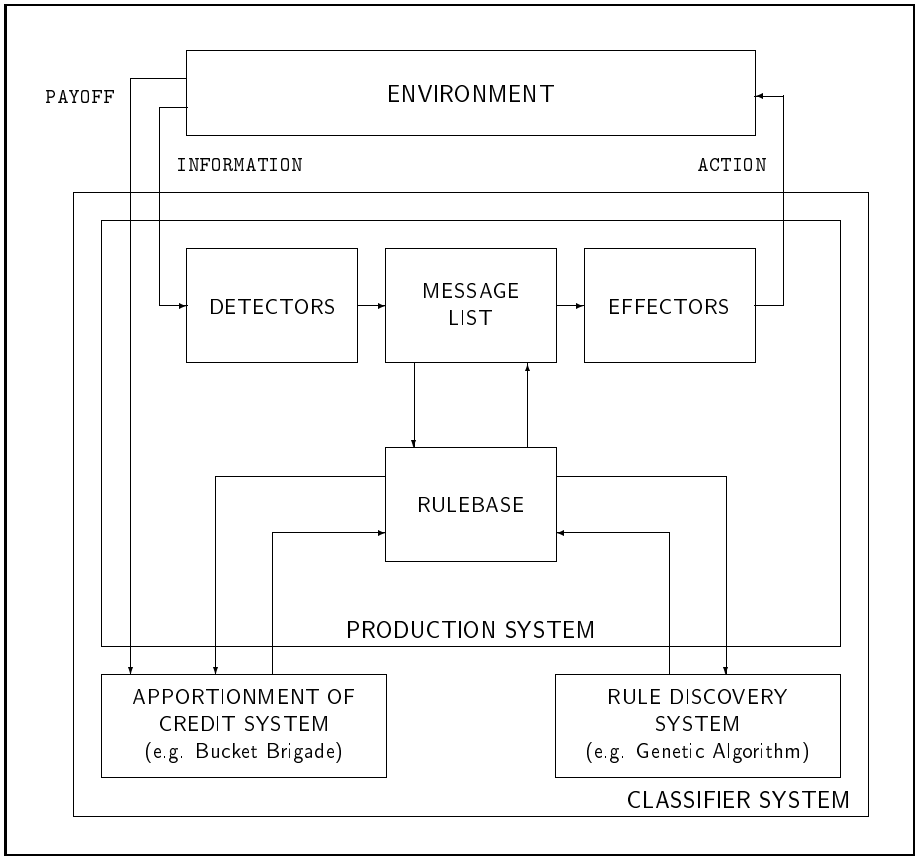


Figure 5.1: A classifier system of the Michigan type

The Production System

First of all, the communication of the production system with the environment is done via an arbitrarily long list of messages. The detectors translate responses from the environment into binary messages and place them on the message list which is then scanned and changed by the rulebase. Finally, the effectors translate output messages into actions on the environment, such as forces or movements.

Messages are binary strings of the same length k. More formally, a message belongs to $\{0,1\}^k$. The rulebase consists of a fixed number m of rules (classifiers) which consist of a fixed number r of conditions and an action, where both conditions and actions are strings of length k over the alphabet $\{0,1,*\}$. The asterisk plays again the role of a wildcard, a "don't care" symbol.

A condition is matched, if and only if there is a message in the list

which matches the condition in all the non-wildcard positions. Moreover, conditions, except the first one, may be negated by adding a "–" prefix. Such a prefixed condition is satisfied, if and only if there is *no* message in the list which matches the string associated with the condition. Finally, a rule fires, if and only if all the conditions are satisfied, i.e. the conditions are connected with AND. Such "firing" rules compete to put their action messages on the message list. This competition will soon be discussed in connection with the apportionment of credit problem.

In the action parts, the wildcard symbols have a different meaning. They take the role of "pass through" element. The output message of a firing rule, whose action part contains a wildcard, is composed from the non-wildcard positions of the action and the message which satisfies the first condition of the classifier (this is actually the reason why negations of the first conditions are not allowed). More formally, the outgoing message $\tilde{m}$ is defined as

$$\tilde{m}[i] := \begin{cases} a[i] & \text{if } a[i] \neq * \\ m[i] & \text{if } a[i] = * \end{cases} \qquad i \in \overline{1,k}, \tag{5.6}$$

where a is the action part of the classifier and m is the message which matches the first condition. Formally, a classifier is a string of the form

$$\text{Cond}_1, [\text{“–”}]\text{Cond}_2, \ldots, [\text{“–”}]\text{Cond}_r/\text{Action}, \tag{5.7}$$

where the brackets should express the optionality of the "–" prefixes.

Moreover, it can be of advantage to supply the messages with prefixes, so-called tags, which identify the origin of the message. Consequently, these prefixes must also be appended to the conditions and actions of the classifiers. In this case we must take special care that no action specifies the prefix reserved for the input interface. This process of tagging offers new opportunities to transfer information about the current step into the next step. This can be accomplished by placing tagged messages on the list which are not interpreted by the output interface. These messages, which, obviously, contain information about the previous step, can support the decisions in the next step. So, appropriate use of tags permits rules to be coupled to act sequentially. In some sense, such messages are the memory of the system.

To summarize this, a single execution cycle of the production system consists of the following steps:

1. Messages from the environment are appended to the message list.
2. All the conditions of all classifiers are checked against the message list to obtain the set of firing rules.
3. The message list is erased.

4. The firing classifiers participate in a competition to place their messages on the list (see below).

5. The winning classifiers place their actions on the list.

6. The messages directed to the effectors are executed.

This procedure is repeated iteratively.

Remark 5.4 How 6. is done, if these messages are deleted or not, and so on, depends on the concrete implementation. It is, on the one hand, possible to choose a representation such that each output message can be interpreted by the effectors. On the other hand, it is possible to direct messages explicitely to the effectors with a special tag. In this case, if no messages are directed to the effectors, the system is in a thinking phase.

If a classifier R_1 produces a message m', which is not directed to the effectors, but tagged as an internal message, and m' satisfies a condition of a classifier R_2 in the next timestep, R_2 is called a consumer of R_1. Reversly, R_1 is called a supplier of R_2.

Credit Assignment — The Bucket Brigade Algorithm

As already mentioned, in each timestep t we assign a strength value $u_{i,t}$ to each classifier R_i. This strength value represents the correctness and importance of a classifier. On the one hand, the strength value influences the chance of a classifier to place its action on the output list. On the other hand, the strength values are used by the rule discovery system which we will soon discuss.

The competition for having the right to post the action together with the adaptation of the strength values depending on the feedback (payoff) from the environment is called the bucket brigade algorithm. It can be regarded as a simulated economic system in which various agents, in our case the classifiers, participate in an auction, where the chance to buy the right to post the action depends on the strength of the agents.

In one of the simpliest forms, the bid of a classifier is defined as

$$b_{i,t} := c_L \cdot u_{i,t} \cdot s_i, \tag{5.8}$$

where $c_L \in [0, 1]$ is a learning parameter, similar to learning rates in artificial neural nets, and s_i is the specifity, the number of non-wildcard symbols in the condition part of the classifier. If c_L is chosen small, the system adapts slowly. If it is chosen too high, the strengths tend to oscillate chaotically.

Then, depending on the bids, the rules, which are allowed to place their output messages on the list, the so-called winning agents, are selected. In the simpliest case this can be done by a random experiment. For each bidding classifier it is decided randomly, if it wins or not, where the probability that it wins is proportional to its bid:

$$\mathsf{P}[r_i \text{ wins}] := \frac{b_{i,t}}{\sum\limits_{j \in \mathrm{Sat}_t} b_{j,t}}, \tag{5.9}$$

where Sat_t is the set of indices which belong to satisfied classifiers at time t.

Obviously, in this approach more than one winning classifiers are allowed. Of course, other selection schemes are reasonable, for instance the highest bidding agent wins alone. This can be necessary to avoid that two winning classifiers direct mutually exclusive actions to the effectors.

Now let us discuss how payment from the environment is distributed and how the strengths are adapted. For this purpose, let us denote the set of classifiers, which have supplied a winning agent r_i in step t, with $S_{i,t}$. Then the new strength of a winning agent is reduced by its bid and increased by its portion of the payoff P_t received from the environment:

$$u_{i,t+1} := u_{i,t} + \frac{P_t}{w_t} - b_{i,t}, \tag{5.10}$$

where w_t is the number of winning agents in the actual time step. A winning agent pays its bid to its suppliers, which share the bid among each other, equally in the simpliest case:

$$u_{l,t+1} := u_{l,t} + \frac{b_{i,t}}{|S_{i,t}|} \qquad \forall r_l \in S_{i,t} \tag{5.11}$$

If a winning agent has also been active in the previous step and supplies another winning agent, the value above is additionally increased by one portion of the bid the consumer offers. In the extreme case, that two winning agents have supplied each other mutually, the portions of the bids are exchanged in the manner as presented above. The strengths of all other classifiers r_n, which are neither winning agents nor suppliers of winning agents, are reduced by a certain factor (they pay a tax):

$$u_{n,t+1} := u_{n,t} \cdot (1 - T), \tag{5.12}$$

where $T \in [0, 1]$ is a small value. The intention of taxation is to punish classifiers which never contribute anything to the output of the system. With this concept redundant classifiers, which never become active, can be filtered out.

The idea behind credit assignment in general and bucket brigade in particular is to increase the strengths of rules which have set the stage for later

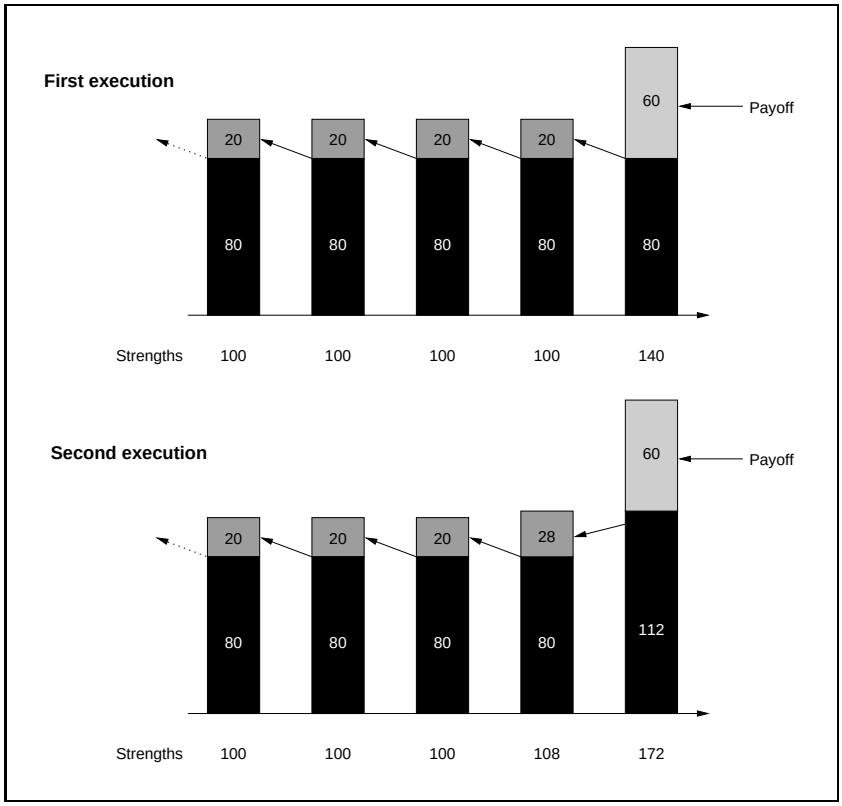


Figure 5.2: The bucket brigade principle

successful actions. The problem of determining such classifiers, which were responsible for conditions under which it was later on possible to receive a high payoff, can be very difficult. Consider for instance the game of chess again, in which very early moves can be significant for a late success or failure. However, the bucket brigade algorithm can solve this problem, although, obviously, strength is only transferred to the suppliers which were active in the previous step. Each time the same sequence is activated, a little bit of the payoff is transferred one step back in the sequence. It is easy to see, that repeated successful execution of a sequence can increase the strengths of all coupled classifiers involved.

Example 5.5 Figure 5.2 shows a simple example how the bucket brigade algorithm works. For simplicity, we consider a sequence of five classifiers which always bid 20 percents of their strength. Only after the fifth step, after the activation of the fifth classifier, a payoff of 60 is received. The further future of this sequence would be the one shown in figure 5.3. It is easy to see from this example that the reinforcement of the strengths is slow

Strength after the					
3rd	100.00	100.00	101.60	120.80	172.00
4th	100.00	100.32	105.44	136.16	197.60
5th	100.06	101.34	111.58	152.54	234.46
6th	100.32	103.39	119.78	168.93	247.57
⋮					
10th	106.56	124.17	164.44	224.84	278.52
⋮					
25th	215.86	253.20	280.36	294.52	299.24
⋮					
execution of the sequence					

Figure 5.3: Repeated bucket brigade

at the beginning but it accelerates later. Exactly this property contributes much to the robustness of classifier system — they tend to be cautious at the beginning, trying not to rush conclusions, but, after a certain number of similar situations, the system adopts the rules more and more. Figure 5.4 shows a graphical visualization of this fact interpreting the table shown in figure 5.3 as a two-dimensional surface.

From example 5.5 and the considerations above it might be clear, that the bucket brigade algorithm fails if the number of environmental states is so large that the system never observes the same sequence more than once.

Rule Generation

While the apportionment of credit system just judges the rules, the purpose of the rule discovery system is to eliminate low-fitted rules and to replace them by hopefully better performing ones. The fitness of a rule is given by its strength. Since the classifiers of a Holland classifier system themselves are strings, the adaptation of a genetic algorithm to the problem of rule induction is straightforward, though many variants are reasonable. Almost all variants have in common that the GA is not invoked in each time step, but only every n-th step, where n has to be set such that enough information about the performance of new classifiers can be obtained in the meantime.

A. Geyer-Schulz, for instance, suggests the following procedure (see [Geyer-Schulz, 1995]):

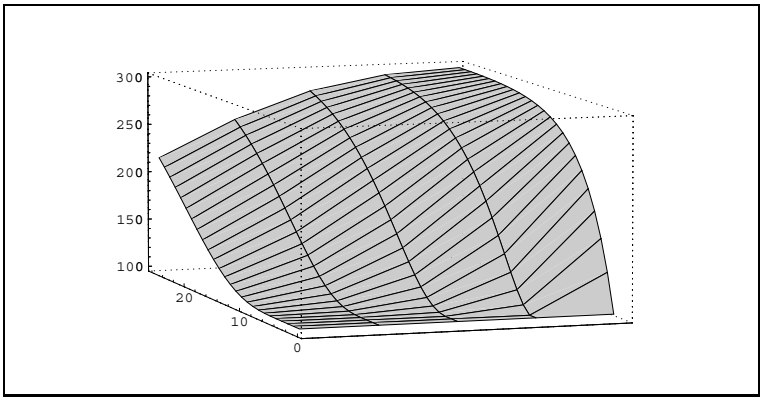


Figure 5.4: A graphical representation of the table shown in figure 5.3

1. select a subpopulation of a certain size at random,
2. take this subpopulation as actual generation and compute a new generation by applying the genetic operations selection, crossing over, and mutation as described in 3.1.2,
3. merge the new subpopulation with the rulebase omitting duplicates and replacing the worst classifiers.

However, this process of acquiring new rules has an interesting side effect. It is more than only the exchange of parts of conditions and actions. Since we have not stated restrictions for manipulating tags, the genetic algorithm can recombine parts of established tags to invent new tags. In the following, tags spawn related tags establishing new couplings. These new tags survive if they contribute to useful interactions. In this sense, the GA additionally creates experience-based internal structures.

Since we have discussed all the important components of a classifier system of the Michigan type in detail, we can now turn to the application of such techniques to the acquisation of fuzzy rules.

5.2.2 Fuzzy Classifier Systems of the Michigan Type

The previous considerations about classifier systems have been made with just the objective in mind to generalize them to the case of online fuzzy-rule-acquisation. While crisp classifier systems of the type discussed above had been introduced by J. H. Holland in 1976, their fuzzification awaited discovery many years. The first papers dealing with that interesting topic

were by M. Valenzuela-Rendón [Valenzuela-Rendón, 1991a] and [Valenzuela-Rendón, 1991b]. Since then not very much has happened. There is at least one other paper by A. Bonarini ([Bonarini, 1993]) dealing with the online learning of incomplete fuzzy rule sets for an autonomous robot. It is not obvious why this field is treated like a stepchild, although the results seem to be encouraging. However, these ideas are really worth being discussed.

We concentrate on M. Valenzuela-Rendón's ideas here. He has introduced a fuzzy classifier system which is, more or less, with little modifications, a generalization of an ordinary Holland classifier system.

The Production System

Here we consider, as M. Valenzuela-Rendón did, a system with real-valued input and output, such as a fuzzy controller. The system has, unlike ordinary fuzzy controllers, three different types of variables — input-, output-, and internal variables. As we will see later, internal variables are for the purpose of storing information about the near past. They correspond to the specially tagged messages in the crisp case. For the sake of generality and simplicity, all the universes of discourse, which are intervals in this case, are linearly transformed to the unit interval $[0, 1]$. For each variable the same number of membership functions n is assumed. These membership functions are fixed at the beginning. They are not changed throughout the learning process. M. Valenzuela-Rendón took bell-shaped function which divided the interval rather equally. Of course, other types can be used, what actually has no effect on our considerations here.

A message is then a binary string of length $l + n$, where n is the number of membership functions defined above and l is the length of the prefix (tag), which identifies the variable to which the message belongs. A good choice for l would be $\log_2 K$, where K is the total number of variables we want to consider. To each message an *activity level*, which represents a truth value, is assigned. Consider for instance the following message ($l = 3$, $n = 5$):

$$\underbrace{010}_{=2} : 00010 \rightarrow 0.6$$

Its meaning is "Input no. 2 belongs to fuzzy set no. 4 with a degree of 0.6". On the message list only so-called minimal messages are used, i.e. messages with only one 1 in the part which identifies the numbers of the fuzzy sets.

Classifiers again consist of a fixed number r of conditions and an action part. Note that, in this approach, no wildcards and no "–" prefixes are used. Both condition and action part are also binary strings of length $l + n$, where the tag and the identifiers of the fuzzy sets are separated by an colon. The degree to which such a condition is matched is then, of course, a fuzzy

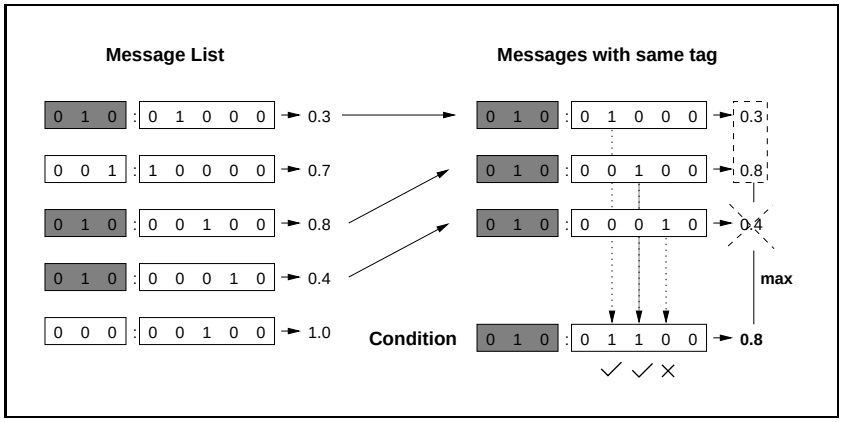


Figure 5.5: Matching a fuzzy condition

truth value between 0 and 1. The degree of matching is computed as the maximal activity of messages on the list, which have the same tag and whose 1s are a subset of those of the condition. Figure 5.5 shows a simple example how this matching is done. The degree of satisfaction of the whole classifier is then computed as the minimum of matching degrees of the conditions. This is then also the activity level which is assigned to the output message (i.e. Mamdani inference with $T_2 = T_{\mathrm{M}}$). This is exactly the same inference method as we have discussed in 5.1.1 (see (5.2) and (5.3) on page 87).

The whole rulebase consists of a fixed number m of such classifiers. Similarly to Holland classifier systems, one execution step of the production system is done as follows:

1. The detectors receive crisp input values from the environment and translate them into minimal messages which are then added to the message list.

2. The degrees of matching are computed for all classifiers.

3. The message list is erased.

4. The output messages of some matched classifiers (see later) are placed on the message list.

5. The output messages are translated into minimal messages. For instance, the message $010 : 00110 \to 0.9$ is split into the two messages $010 : 00010 \to 0.9$ and $010 : 00100 \to 0.9$.

6. The effectors discard the output messages (referring to output variables) from the list and translate them into instructions to the environment.

From point 2 it can be seen easily that it is of advantage to use fuzzy sets with local support instead of bell-shaped ones, because, if bell-shaped fuzzy sets are used, each rule fires in each time step.

Step 6 is done by a modified Mamdani inference: The sum (instead of the maximum or another t-conorm) of activity levels of messages, which refer to the same fuzzy set of a variable, is computed. The membership functions are then scaled with these sums. Finally, the center of gravity of the "union" (i.e. maximum) of these functions, which belong to one variable, is computed.

Credit Assignment

Since fuzzy systems have been designed to model transitions, a probabilistic auction process as discussed in connection with Holland classifier systems, where only a small number of rules is allowed to fire, is not desirable. Of course, we again assign strength values to the classifiers.

If we are dealing with a one-stage system, in which payoff for a certain action is received immediately, where no long-term strategies must be evolved, we can suffice with allowing all matched rules to post their outputs and sharing the payoff among the rules, which were active in the last step, according to their activity levels in this step. For example, if R_t is the set of classifiers, which have been active at time t, and P_t is the payoff received after the t-th step, the modification of the strengths of firing rules can be defined as

$$u_{i,t+1} := u_{i,t} + P_t \cdot \frac{a_{i,t}}{\sum\limits_{r_i \in R_t} a_{i,t}} \qquad \forall r_i \in R_t, \tag{5.13}$$

where $a_{i,t}$ denotes the activity level of the classifier r_i at time t. It is again possible to reduce the strength of inactive classifiers by a certain tax.

In the case, that the problem is so complex that long-term strategies are indispensable, a fuzzification of the bucket brigade mechanism must be found. While Valenzuela-Rendón only provides a few vague ideas, we state one possible variant, where the firing rules pay a certain value to their suppliers which depends on the activity level. The strength of a classifier, which has recently been active in time step t is then increased by a portion of the payoff as defined in (5.13), but it is additionally decreased by a value

$$b_{i,t} := c_L \cdot u_{i,t} \cdot a_{i,t}, \tag{5.14}$$

where $c_L \in [0, 1]$ is again the learning rate. Of course, it is again possible to incorporate terms which depend on the specifity of the classifier.

This "bid" is then shared among the suppliers of such a firing classifier according to the amount they have contributed to the matching of the consumer. If we consider an arbitrary but fixed classifier r_j which has been

active in step t and if we denote the set of classifiers supplying r_j, which have been active in step $t-1$, with $S_{j,t}$, the change of the strengths of these suppliers can be defined as

$$u_{k,t+1} := u_{k,t} + b_{j,t} \cdot \frac{a_{k,t-1}}{\sum_{r \in S_{j,t}} a_{r,t-1}} \qquad \forall r_k \in S_{j,t}. \tag{5.15}$$

It is easy to see, that this can be an appropriate generalization of the bucket brigade algorithm as described in 5.2.1.

Rule Discovery

The adaptation of a genetic algorithm to the problem of manipulating classifiers in our system is again straightforward. We only have to take special care that tags in conditional parts must not refer to output variables and that tags in the action parts of the classifiers must not refer to input variables of the system.

Analogously to the considerations in 5.2.1, if we allow a certain number of internal variables, the system tends to build internal chains, coupled sequences, autonomously. If we allow internal variables, a classifier system of this type not only learns stupid input-output actions, it also tries to discover causal interrelations.

Bonarini's Modifications

A. Bonarini's paper [Bonarini, 1993] is more application-oriented. Not so much about the details inside is revealed. However, he concentrates on finding a small rulebase which only contains important rules. In this approach the number of rules is allowed to vary between certain boundaries.

The number of rules can be varied in each time step depending on the number of rules which match the actual situation. This is done by two dual operations:

1. If the rules, which match the actual situation, are too many, the worst of them is deleted.

2. If there are too few rules matching the current inputs, a new rule, whose antecents cover the current state, with randomly chosen consequent value, is added to the rulebase.

The genetic operations are only applied to the consequent values of the rules. Since the antecedents are generated on demand in the different timesteps, no taxation is necessary. Bonarini has called this mechanism "cover-detector algorithm".

5.3 Conclusion

While the acquisation of decision tables as briefly discussed in 5.1.1 is, at least from the mathematical point of view, a rather boring task, fuzzy genetic programming and the theory of fuzzy classifier systems of the Michigan type are very challenging fields which are both still in their infancy.

The slowly increasing number of publications, which deal with these topics, might indicate that they are continuously approaching a promising future. The first challenge is to apply them to more complex problems. All practical results, which have been published until now, concern with the applications to comparatively simple examples. Furthermore, the combination of these two paradigms to design systems, which learn program-like structures in an online process, could open a completely new, revolutionary area.

Chapter 6

More about the Combination of Evolutionary and Fuzzy Methods

> *As long as algebra and geometry proceeded along separate paths, their advance was slow and their applications limited. But when these sciences joined company they drew from each other fresh vitality and thenceforward marched on at a rapid pace towards perfection.*
>
> Joseph Louis Lagrange

Of course, one can be of different opinion about the synergism of algebra and geometry, but, on the contrary, it is undoubted that, whenever two theories are joined together, both can profit from the other. In some sense, this also applies to the combination of two distinct groups of practical methods which can, although they are applied in completely different fields, support the usefulness and applicability of the respective other one.

In the last two chapters we have discussed two main streams of combining fuzzy logic and genetic algorithms. Now it is time to conclude this thesis with a view to other approaches to the combination of these two paradigms. First, we give a brief overview of methods for optimizing whole controllers, where, in some sense, fuzzy sets and "rules" are optimized simultaneously. The second section will deal with a completely different aspect which we have disregarded until now — the optimization of the hierarchical structure of a fuzzy system.

6.1 Optimizing Whole Controllers

6.1.1 The Ideas of Lee and Takagi

The first approach has been introduced by M. A. Lee and H. Takagi and can be found in [Lee and Takagi, 1993c] and [Takagi and Lee, 1993]. Takagi and Lee have applied a genetic algorithm to the optimization of a Takagi-Sugeno controller (note: different Takagi!) which consists of a finite number of rules of the form

$$\mathsf{IF} \quad x \text{ is } A_j \quad \mathsf{THEN} \quad y \text{ is } f_j(x), \tag{6.1}$$

where f_j is an affine linear mapping (see 2.5.2).

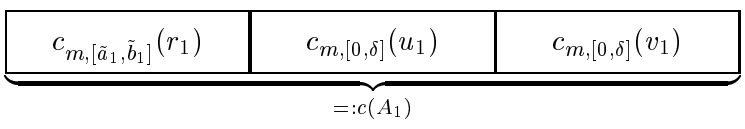
If the universe of discourse is two-dimensional and if we assume that the rulebase is consistent and complete, what actually means that each premise occurs exactly once in the rulebase, (6.1) can be rewritten as

$$\mathsf{IF} \quad x_1 \text{ is } A_i \text{ AND } x_2 \text{ is } B_j \quad \mathsf{THEN} \quad y \text{ is } \alpha_{ij} \cdot x_1 + \beta_{ij} \cdot x_2 + \gamma_{ij}, \tag{6.2}$$

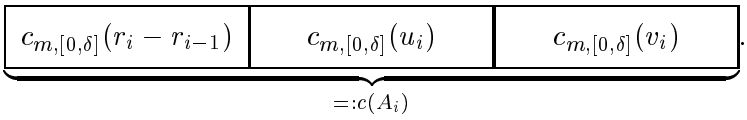
where A_i and B_j are fuzzy subsets of $[a_1, b_1]$ and $[a_2, b_2]$, the universes of discourse of the fuzzy variables x_1 and x_2, respectively. The index i ranges between 1 and N_1, the number of fuzzy subsets specified for the fuzzy variable x_1 (see 2.16 for comparison). Analogously, j ranges between 1 and N_2, the number of fuzzy subsets of the fuzzy variable x_2. Such a rulebase can be expressed uniquely with a matrix of the form (compare with the decision table in 5.1.1)

	B_1	$\cdots$	B_{N_2}
A_1	$\begin{pmatrix} b_{11} \\ c_{11} \\ d_{11} \end{pmatrix}$	$\cdots$	$\begin{pmatrix} b_{1N_2} \\ c_{1N_2} \\ d_{1N_2} \end{pmatrix}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
A_{N_1}	$\begin{pmatrix} b_{N_1 1} \\ c_{N_1 1} \\ d_{N_1 1} \end{pmatrix}$	$\cdots$	$\begin{pmatrix} b_{N_1 N_2} \\ c_{N_1 N_2} \\ d_{N_1 N_2} \end{pmatrix}$.

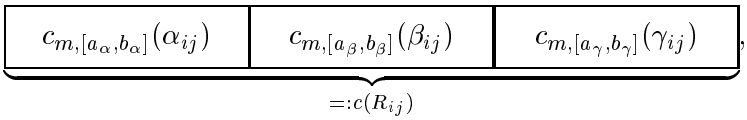
In this approach triangular fuzzy sets were used. For the fuzzy subsets $A_1, \ldots, A_{N_1}$ and $B_1, \ldots, B_{N_2}$ fuzzy partitions as shown in 4.1.2 could be reasonable. Here we have more degrees of freedom, but without allowing arbitrary configurations which violate a natural ordering of the fuzzy sets. First of all, the triangular set A_1 (we only demonstrate that for the first variable) is encoded as already discussed on page 59:



The other sets A_i, whose medians must appear in increasing order, are encoded as

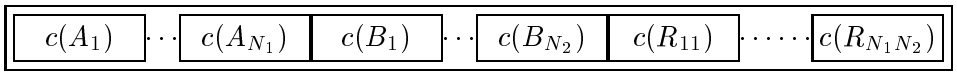


The consequent parameters are encoded as



where the intervals $[a_\alpha, b_\alpha]$, $[a_\beta, b_\beta]$, and $[a_\gamma, b_\gamma]$ must be chosen depending on the needs of the concrete application.

The whole controller can then be encoded by putting the codings of the fuzzy sets and the codings of the consequent parameters into one string:



Obviously, it is determined uniquely by the positions in the string which consequent parameters belong to which premises.

Takagi and Lee applied their ideas, which have been discussed in a more general form here, to the problem of balancing an inverted pendulum, a well-known example in the world of fuzzy control. They used two-point crossing over with a probability of 0.6 and standard mutation with a probability of 0.0333. The size of the population was 10. They have reported sufficiently good results. They also tried out various methods to incorporate a priori knowledge, such as symmetries, with the intention to reduce the size of the search space or to support the GA in its search for the optimum. The fitness measure they used was the number of steps it took to fulfill a certain balance criterion. In the case that the pendulum fell over a certain high value was taken as payoff.

The generalization of the techniques presented above to the case of controllers with more than two inputs is, apparently, straightforward. It is easy to see that the length of the strings depends exponentially on the dimension.

6.1.2 The Nagoya Approach

The second idea goes back to T. Furuhashi, K. Nakaoka, and Y. Uchikawa from the university of Nagoya, Japan. In their approach a modified kind of

genetic algorithm is used. Pretentiously, they have called their idea "Nagoya Approach". It can be found in [Furuhashi *et al.*, 1995].

The three authors applied a genetic algorithm to the optimization of a Mamdani controller for guiding a robot to a certain goal through a room which contains a moving obstacle. This controller had five inputs (compare with figure 6.1)

D	...	actual distance from the obstacle,
Φ	...	angle between the robot and the relative velocity v_R between the robot and the obstacle,
δ	...	angle between the actual path of the robot and the obstacle,
Ψ	...	angle between the actual path and the goal,
ω	...	angular velocity of the robot,

and two outputs

u	...	steering angle of the robot,
Δv	...	change of the absolute value of the speed of the robot.

For the input variables they used bell-shaped fuzzy sets; the sets were additionally scaled with a factor $h \in (0, 1]$ (see also page 60):

$$x \longmapsto h \cdot e^{-\frac{(x-r)^2}{2u^2}} \tag{6.3}$$

It is near at hand to use the following coding for sets of this shape:

$c_{m,[\varepsilon,1]}(h_v)$	$c_{m,[a,b]}(r_v)$	$c_{m,[\varepsilon,\delta]}(u_v)$

For the output variables ordinary triangular membership functions were used with the modification that only one offset was used for both left and right. Of course, an appropriate coding is (see also 4.1.1)

$c_{m,[a,b]}(r_v)$	$c_{m,[\varepsilon,\delta]}(u_v)$

.

In this approach it was assumed that the whole system is controllable with a certain fixed number N (concretely 15 in this application) of rules of the shape

IF $\tilde{T}_i(D$ is $A_{D,i},\ \Phi$ is $A_{\Phi,i},\ \delta$ is $A_{\delta,i},\ \Psi$ is $A_{\Psi,i},\ \omega$ is $A_{\omega,i})$ THEN
u is $A_{u,i}$ AND Δv is $A_{\Delta v,i}$,

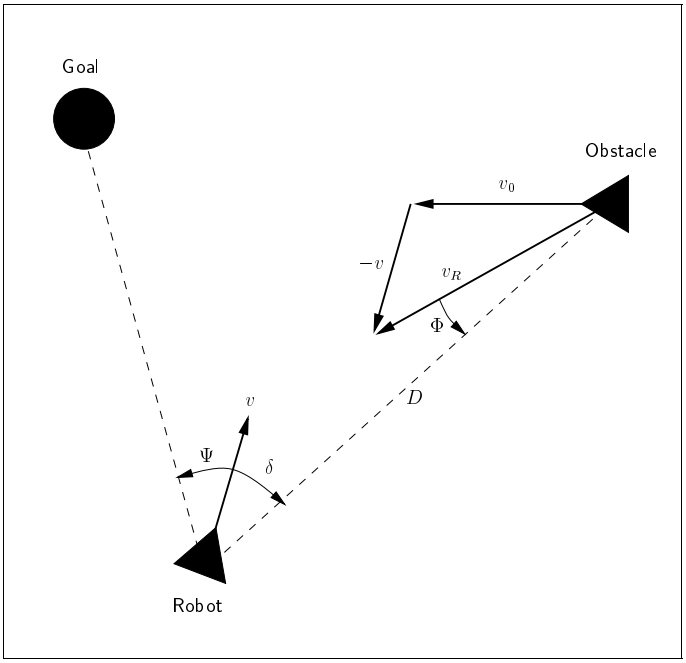
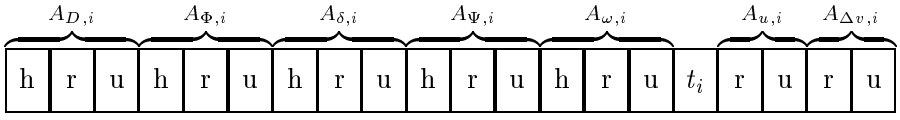


Figure 6.1: Sketch of the obstacle avoidance problem

where $\tilde{T}_i$ is an operator which lies between product and the sum, i.e.

$$\tilde{T}_i(x, y) := t_i \cdot x \cdot y + (1 - t_i) \cdot (x + y) \quad \text{with } t_i \in [0, 1]. \tag{6.4}$$

The output of such a rule is computed by deriving the "truth" values, which need not be lower or equal than 1, of the premises using the operator $\tilde{T}_i$ and by applying a modified Mamdani inference (actually the same as in 5.2.2 on page 102) for both output variables independently. A rule of this kind can be encoded as



Apparently, in this approach all three sets of parameters are tried to be optimized, the fuzzy sets, the rules, and the operations as well.

The most interesting novelty of this paper is that a new modification of an ordinary genetic algorithm is introduced. It provides a better local improvement of single rules. The idea is the following: The whole individual is divided into N_p parts which can be judged independently. For each individual of the actual population a certain number M of clones of each

part are produced. $M-1$ of them are mutated. The best clone of each part is then implanted into the new individual. After this step, normal selection and crossing over are performed. Normally, mutation yields bad individuals very often, because one modification can deteriorate the fitness of the whole individual. In this approach various local phenomena are judged independently. The positive effect is a better local improvement of the individuals. The following algorithm gives a more detailed view of this procedure:

Algorithm 6.1 Let f_i denote the local fitness function which judges the local fitness of part no. i; $g[i]$ denotes the i-th part of an individual g.

$t := 0$;
Compute initial population $\mathcal{B}_0 = (b_{1,0}, \ldots, b_{m,0})$;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
  FOR $i := 1$ **TO** m **DO**
  BEGIN

    FOR $j := 1$ **TO** N_p **DO**
    BEGIN

(* produce M clones and mutate them *)

      $Clone_1 := b_{i,t}[j]$;
      FOR $k := 2$ **TO** M **DO**
      BEGIN
        $Clone_k := b_{i,t}[j]$;
        mutate $Clone_k$
      END

(* implant the best-fitted one into the individual *)

      determine $Clone_l$ *with highest fitness* $f_i(Clone_k)$;
      $b_{i,t}[j] := Clone_l$
    END
  END

(* perform selection and crossing over as usual *)

  FOR $i := 1$ **TO** m **DO**
  BEGIN
    select an individual g *from* $\mathcal{B}_t$;

    IF Random$[0,1] \leq p_{\mathbf{C}}$ **THEN**
      cross g *with a randomly chosen individual of* $\mathcal{B}_t$;

    $b_{i,t+1} := g$
  END $t := t + 1$
END

The 15 rules were encoded by putting the coding presented above into

one long string — three parts with five rules in each case. Each part was evaluated under five different simulation conditions. In the case of success, it received the payoff

$$1 + \frac{100}{\text{No. of steps}}.$$

In the case of failure, the payoff was

$$\frac{10}{\text{Distance to the goal}}.$$

The simulation space was a rectangular room with size 640×750. The fitness of the whole individual was computed as the sum of the local payoffs of the three parts. The technique shown above was applied with a population size of 10, proportional selection, and one-point crossing over. The mutation of the clones of the parts was done by selecting a parameter randomly and by replacing this parameter by a uniform random number.

The authors have reported that the best controller of the 20-th generation was able to guide the robot successfully to the goal under all the five simulation conditions. They compared that with a certain Pitt approach. It was not even able to yield a set of rules, which was able to perform that well, after 275 generations and an amount of computations, which was two times larger.

6.1.3 An Approach with Variable Size of the Rulebase

Now we briefly discuss an approach in which not only the parameters of a fixed number of rules, but also the size and complexity of the rulebase are tuned. These ideas were published by T. Fukuda, Y. Hasegawa, and K. Shimojima in [Fukuda *et al.*, 1995] and [Shimojima *et al.*, 1995].

In their approach a Sugeno controller with n inputs consisting of a certain number of rules of the kind

$$\mathsf{IF} \quad x_1 \ \mathsf{is} \ A_{1i} \ \mathsf{AND} \ \cdots \ \mathsf{AND} \ x_n \ \mathsf{is} \ Ani \quad \mathsf{THEN} \quad y \ \mathsf{is} \ b_i \tag{6.5}$$

is considered, where $A_{j,i}$ are radial basis fuzzy sets and b_i are real values. One such rule is encoded as

The codings $c(A_{ji})$ are the same as proposed in 4.1.1 on page 61. The values e_{ji} are single bits, the so-called "enable"-bits. If such a bit is 1, the membership function which follows the bit is taken into account in the evaluation of the premise, otherwise this position is considered as a wildcard,

what actually means, that the rule does not care about the value of this variable. If all the enable-bits of a rule are 0, the rule is not evaluated. Obviously, these enable-bits determine the number of rules and the number of different membership functions. The input-output function of such a controller is given by (compare with definition 2.31)

$$\begin{array}{rccl} Y: & \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \vec{x} = (x_1, \ldots, x_n) & \longmapsto & Y(\vec{x}) := \dfrac{\sum\limits_{i=1}^{N} \mu_i(\vec{x}) \cdot b_i}{\sum\limits_{i=1}^{N} \mu_i(\vec{x})}, \end{array} \tag{6.6}$$

where the values

$$\mu_i(\vec{x}) := \begin{cases} 0 & \text{if all } e_{1i}, \ldots, e_{ni} \text{ are } 0 \\ \prod\limits_{j=1}^{n} \left(1 - e_{ji} \cdot \left(1 - \mu_{A_{ji}}(x_j)\right)\right) & \text{otherwise} \end{cases}$$

denote the truth values of the premises and N is the total number of rules. It is easy to see that this function is well-defined, whenever at least one rule is active; otherwise, the output of the system is set fo 0 in advance.

The enable-bits are intended to allow generalizing rules which can decrease the size of a rulebase remarkably. Moreover, since we have a clever coding for such a controller, which contains generalizing rules, it is possible to incorporate the number of rules and membership functions in the fitness function. Then the process of finding an optimal rulebase, which has, in addition, a low complexity, can be carried out by a genetic algorithm. Indeed, the authors have proposed

$$F := \alpha \cdot E_{\mathbf{avg}} + \beta \cdot E_{\mathbf{max}} + \gamma \cdot N_R + \delta \cdot N_M \tag{6.7}$$

as appropriate fitness measure, where

$$E_{\mathbf{avg}} := \frac{1}{K} \sum_{k=1}^{K} \left(Y(\vec{x}_k) - y_k\right)^2$$

is the average square distance between actually obtained outputs $Y(\vec{x}_k)$ and desired output y_k for the input values $\vec{x}_k$.

$$E_{\mathbf{max}} := \max_{k=1}^{K} \left(Y(\vec{x}_k) - y_k\right)^2$$

is the maximal square distance. N_R is the number of active rules and N_M is the number of active membership functions; α, β, γ, and δ are coefficients, which control the priority of the four quality measures. For example, if γ was very high, the genetic algorithm would tend to find small rulebases rather than to take much care of the correctness of the output.

If we fix a crossing over method and a mutation operation, we can apply an ordinary genetic algorithm to the optimization problem above. Hasegawa, Fukuda, and Shimojima used two-point crossing over and standard mutation. However, they have additionally introduced a gradient method in order to support the genetic algorithm.

It is easy to see that the membership functions are differentiable. Since the input-output function (6.6) is just the composition of differentiable functions, it is differentiable too. Furthermore, the output for a fixed $\vec{x}$ depends differentiably on the parameters, which determine the shape of the radial basis sets, and on the consequent parameters b_i. Thus, the error measure

$$E := \frac{1}{2} \sum_{k=1}^{K} \left(Y(\vec{x}_k) - y_k \right)^2 , \tag{6.8}$$

where $\vec{x}_k$ and y_k are defined as above, depends differentiably upon these parameters. So, it is possible to formulate a conventional gradient method for minimizing (6.8). One step of such a gradient method (without line search) is given as

$$\left.\begin{array}{l} r'_{A_{ij}} := r_{A_{ij}} - k_r \cdot \frac{\partial E}{\partial r_{A_{ij}}} \\ q'_{A_{ij}} := q_{A_{ij}} - k_q \cdot \frac{\partial E}{\partial q_{A_{ij}}} \\ u'_{A_{ij}} := u_{A_{ij}} - k_u \cdot \frac{\partial E}{\partial u_{A_{ij}}} \end{array}\right\} \quad \begin{array}{l} \text{for all } i = 1, \ldots, N \text{ and} \\ \text{for all } j = 1, \ldots, n \end{array} \tag{6.9}$$

$$b'_i := b_i - k_b \cdot \frac{\partial E}{\partial b_i} \qquad \text{for all } i = 1, \ldots, N,$$

where the values $r_{A_{ij}}$, $q_{A_{ij}}$, $u_{A_{ij}}$, and b_i denote the previous iterations and the values $r'_{A_{ij}}$, $q'_{A_{ij}}$, $u'_{A_{ij}}$, and b'_i denote the new iterations; k_r, k_q, k_u, and k_b are again learning parameters. Fukuda, Hasegawa, and Shimojima have proposed a hybrid method in which the GA is additionally supported by the gradient method above.

Summarized, the optimization of the whole controller is done by

Algorithm 6.2

Compute initial population;

WHILE *stopping condition not fulfilled* **DO**
BEGIN
perform one step (6.9) for each individual;
compute new generation as usual;
END

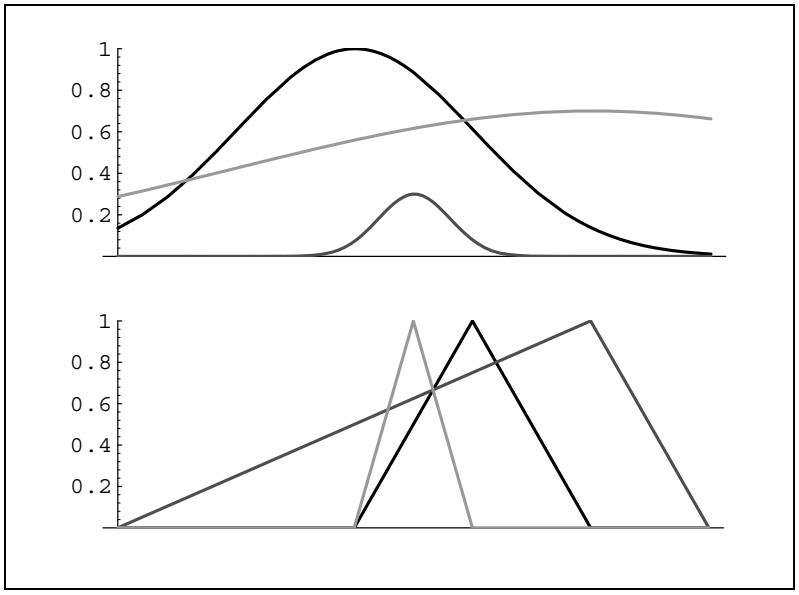


Figure 6.2: Difficultly interpretable configurations of fuzzy sets

This method can be used for the optimization of any Sugeno controller of the type presented above. In the case of this approach the method was incorporated in an algorithm which additionally optimizes the hierarchical structure of a fuzzy system. This approach merits being discussed in its own right, which will be done in 6.2.

6.1.4 Critique

One of the most important advantages of fuzzy systems is that the functions are parameterized in a way which can be interpreted linguistically. More precisely, it is possible to translate human knowledge into fuzzy rules and fuzzy sets, but, on the contrary, not every reasonable fuzzy system is easily interpretable for humans. Consider for instance the configurations shown in figure 6.2. In none of the approaches, which we have discussed above, cases like these are excluded. Moreover, the probability, that such difficultly interpretable configurations are obtained, is rather high. An alternative, which can help to overcome this problem, is to use fuzzy partitions as presented in 4.1.2. Obviously, this approach allows less degrees of freedom, what can reduce the size of the search space and, therefore, speed up the convergence.

We have seen that the methods for encoding fuzzy controllers are, more or less, straightforward. Not so surprisingly, a lot of similar ideas have been

published. An incomplete list of such papers would be [Castro *et al.*, 1993], [Mitsubuchi *et al.*, 1993], [Surmann *et al.*, 1993], [Kacprzyk, 1995], [Lin and Chen, 1995], [Magdalena and Monasterio, 1995], and [Yubazaki *et al.*, 1995].

6.2 Optimizing Hierarchical Structures of Fuzzy Systems

Consider for instance a fuzzy system with 14 inputs, each with three verbal values with which we associate fuzzy sets, and one output with five verbal values. Then the total number of different premises, which specify each variable in the premise, is $3^{14} = 4782969$ and the total number of rules with premises of such a kind is $5 \cdot 4782969 = 23914845$. Obviously, this is a size which is not so easy to survey. This complexity entails the necessity to use either generalizing rules or to prepare a hierarchical structure, which bundles the information such that the decisions are divided up into a certain number of subdecisions.

In all our previous considerations we have either dealt with fuzzy controllers without a hierarchical structure or we have assumed that the structure of a fuzzy system is fixed. In many applications, where the relationships between the different sets of data are unknown, the preparation of an appropriate hierarchy is a very difficult task. Of course, it is desireable to have methods which can help to find such a hierarchy. However, there is at least one paper ([Shimojima *et al.*, 1995]) by T. Fukuda, Y. Hasegawa, and K. Shimojima, which we have already discussed in a different context, dealing with the acquisation of an optimal hierarchical structure applying genetic algorithms. By the way, the same results have been published in [Fukuda *et al.*, 1995].

These four authors have presented a coding for hierarchies which can then be incorporated in a genetic optimization process which tunes the structure too. This coding only applies to tree-like structures. Starting point is a binary tree which consists of a certain number of units (i.e. rulebases). Fukuda, Hasegawa, and Shimojima have recommended 2^{n-2}, where n is the number of input variables of the whole fuzzy system. Then the numbers of the units, to which the input variables are connected, are encoded. We only consider units which are connected to input variables and the units above. All the others are removed. If there are still units, which have exactly one input variable as the one and only input, these units are also removed. The afore said input variables are then connected to the unit above. Figure 6.3 shows an example for six inputs which illustrates the coding and how the structure is simplified. Not that the decoding function is not injective in this case, different genotypes can have the same phenotypes.

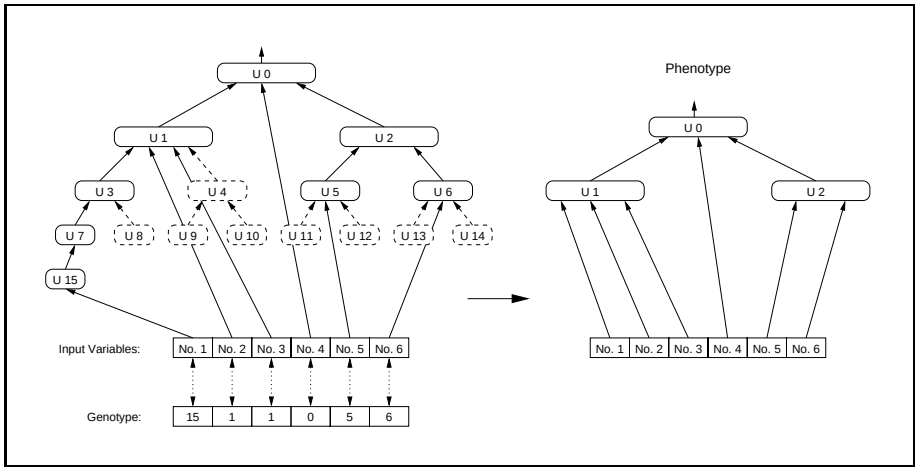


Figure 6.3: Example for coding a hierarchical structure

To this coding all the conventional crossing over and mutation techniques can be applied without modifications. Fukuda, Hasegawa, and Shimojima have used elementwise two-point crossing over, where one element is a binary string of length $n-2$. The value of the i-th element refers to the unit to which input variable no. i is connected. For mutating such a hierarchy they have selected one element randomly and replaced the number by a randomly chosen one. Figure 6.4 shows an example how two hierarchies are crossed with elementwise two-point crossing over.

The concept of coding a hierarchy can now be incorporated in a optimization process. One possibility, where the hierarchy is tuned before the rulebases, can be found in [Shimojima *et al.*, 1995]. The process of tuning the hierarchical structure can be outlined as follows:

Algorithm 6.3

Initialize variables; [1]
Choose initial generation; [2]

WHILE *stopping condition not fulfilled* **DO**
BEGIN
tune consequent values; [3]
compute next generation; [4]
END

[1] The complete tree is prepared. For each variable (both input variables and intermediate ones which transmit information from one unit to the next

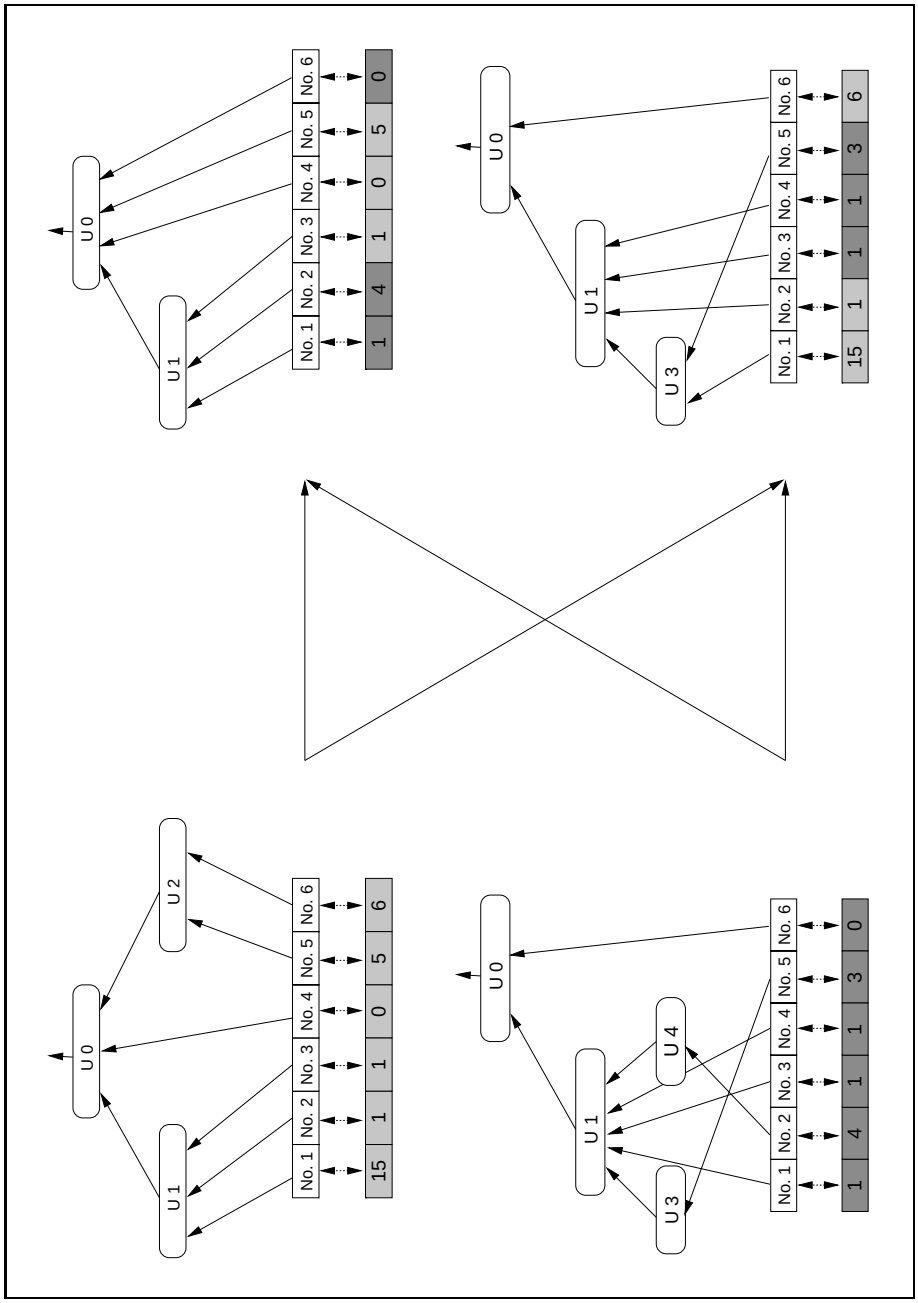


Figure 6.4: Example for crossing two hierarchical structures

one), a certain number of membership functions (e.g. 3) is assumed. Initially, they are set equal for all variables, such that they divide the input space rather equally.

[2] Hierarchies are chosen randomly. The consequent parameters of the rules are initially set to 0.

[3] The consequent values are tuned for all units from the top to the bottom with the method shown in (6.9). For this purpose it is necessary to know the desired output values of all the intermediate units, which is nearly impossible. In this approach the desired outputs of intermediate units are computed with the back-propagation method (BP): The desired output Y_1 of a unit U_1, which is the k-th input of a unit U_2, whose desired output Y_2 is known, is approximated by

$$Y_1 = Y_2 - \frac{\partial E_2}{\partial x_k}, \tag{6.10}$$

where E_2 is the square error between the desired output and the output actually obtained by the evaluation of U_2. By the way, this methods comes from the theory of artificial neural nets (see [Rumelhart and McClelland, 1986] or [Zurada, 1992] for reference).

[4] In this step, normal selection, crossing over, and mutation are applied to the population of hierarchies. The function (6.7) is used as fitness function. The numbers of rules and membership functions are computed as the sums over all units.

Apparently, this is only a raw search for the optimal fuzzy system, because the fuzzy sets are fixed, but it is a pretty good method for acquiring a fairly good hierarchical structure. In order to optimize all the parameters, when a good hierachy had been found already, the authors have applied the methods presented in 6.1.3 to the hierarchy found by the algorithm above.

Of course, other methods for incorporating the search for an optimal hierarchical structure in a process of automatically modeling a fuzzy system are reasonable. For instance, it could be of advantage to tune all the parameters, including the structure, simultaneously. Obviously, this is a very new field of research with a lot of open questions.

6.3 Conclusion

This chapter has demonstrated that the field of combining genetic algorithms with fuzzy systems is a very broad one, the possibilities of which are not yet exhausted. On the other hand, section 6.1 has shown how hetergeneous this field is and how the approaches, which have been published within the last years, differ.

Appendix A

Mathematical Preliminaries

A.1 Symbols and Notations Used in This Text

This section provides definitions of commonly used symbols and notations which every reader should know. Since there are still a lot of different notations for exactly the same things going around, the purpose of these lines is to define them exactly in order to avoid misunderstandings.

A.1.1 Sets, Functions

$\mathsf{C}A$	complement of set A
$P(A)$	powerset (set of all subsets) of A
$[a, b]$	closed interval
(a, b)	open interval
$(a, b]$, $[a, b)$	half-open intervals
$\overline{1, n}$	compact notation for the set $\{1, \ldots, n\}$
X^Y	set of all functions from X to Y
$\|A\|$	cardinality (number of elements) of set A
$f(A)$	image (range) of set $A \subseteq X$ with respect to function $f : X \to Y$; $f(A) := \{y \in Y \| \exists x \in A : f(x) = y\}$
$f^{-1}(B)$	inverse image of subset $B \subseteq Y$ with respect to function $f : X \to Y$; $f^{-1}(B) := \{x \in X \| f(x) \in B\}$
$\mathrm{Im}(f)$	image (range) of function $f : X \to Y$; $\mathrm{Im}(f) := f(X)$
χ_M	characteristic function of set $M \subseteq X$; see chapter 2 for the exact definition.

A.1.2 Metrics, Norms

Definition A.1 A function

$$\begin{array}{rccc} d: & \Omega \times \Omega & \longrightarrow & \mathbb{R}^+ \\ & (x,y) & \longmapsto & d(x,y) \end{array}$$

is called a pseudometric on Ω if and only if it fulfills the following three conditions:

1. $\forall x, y \in \Omega: \quad x = y \Longrightarrow d(x,y) = 0,$
2. $\forall x, y \in \Omega: \quad d(x,y) = d(y,x),$
3. $\forall x, y, z \in \Omega: \quad d(x,z) \leq d(x,y) + d(y,z).$

If 1. is replaced by the stronger condition

$$\forall x, y \in \Omega: \quad x = y \Longleftrightarrow d(x,y) = 0,$$

it is called a metric on Ω.

Definition A.2 Let Ω be an arbitrary linear vector-space over $\mathbb{R}$. Then a mapping $\|.\| : \Omega \to \mathbb{R}^+$ is called a norm on Ω if and only if

1. $\forall x \in \Omega: \quad x = 0 \Longleftrightarrow \|x\| = 0,$
2. $\forall \lambda \in \mathbb{R}\ \forall x \in \Omega: \quad \|\lambda x\| = |\lambda| \|x\|,$
3. $\forall x, y \in \Omega: \quad \|x + y\| \leq \|x\| + \|y\|.$

Definition A.3 For a given function $f : \Omega \subseteq \mathbb{R} \to \mathbb{R}$ the L_p-norm (if the integral exists) is defined by

$$\|f\|_p := \begin{cases} \left(\int\limits_\Omega |f(x)|^p \, dx \right)^{\frac{1}{p}} & \text{if } p \in [1, \infty) \\ \operatorname{ess\,sup}\limits_{x \in \Omega} |f(x)| & \text{if } p = \infty. \end{cases} \tag{A.1}$$

For a given sequence $(x_n)_{n \in \mathbb{N}}$ the l_p-norm (under the assumption that the sum exists) is defined by

$$\|(x_n)_{n \in \mathbb{N}}\|_p := \begin{cases} \left(\sum\limits_{n=1}^{\infty} |x_n|^p \right)^{\frac{1}{p}} & \text{if } p \in [1, \infty) \\ \sup\limits_{n \in \mathbb{N}} |x_n| & \text{if } p = \infty. \end{cases} \tag{A.2}$$

This definition can also be applied to real vectors $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ with obvious modifications.

A.2 Basic Elements of Probability Theory

This section describes the basic elements of probability theory as they are used, in particular, in chapter 3. Also some important results, which are used in proofs of certain theorems in this thesis, are cited. This is not intended to be a full introduction to probability theory. Further details and proofs are provided e.g. in [Feller, 1968], [Feller, 1971], and [Parzen, 1960].

Definition A.4 Let the set Ω be the sample description space of a random phenomenon. Then subsets $A \subseteq \Omega$ are called events. One-elementary subsets $\{\omega\}$ are called elementary events.

Definition A.5 A system $\mathcal{A}$ of subsets of Ω is called σ-algebra over Ω if and only if

1. $\Omega \in \mathcal{A}$
2. For every $A, B \in \mathcal{A}$ also $A \backslash B \in \mathcal{A}$
3. Every countable union of subsets, which are contained in $\mathcal{A}$, is also in A.

Remark A.6 For $\Omega = \mathbb{R}$ the so-called system $\mathcal{B}$ of Borel-measurable sets is commonly used. It is defined as the smallest σ-algebra which contains all intervals. It can be regarded as the σ-algebraic closure of the set of intervals.

Definition A.7 Assume that $\mathcal{A}$ is a σ-algebra over Ω. Then a mapping $\mathsf{P} : \mathcal{A} \to [0, 1]$ is called a probability measure over $\mathcal{A}$ if and only if $\mathsf{P}[\Omega] = 1$ and the equation

$$\mathsf{P}[\bigcup_{n=1}^{\infty} A_n] = \sum_{n=1}^{\infty} \mathsf{P}[A_n] \tag{A.3}$$

holds for pairwise disjoint subsets $(A_n)_{n \in \mathbb{N}}$ of $\mathcal{A}$. We denote $\mathsf{P}[A]$ with "probability of event A". Events A, B with $A \cap B = \emptyset$ are called mutually exclusive events. A triple $(\Omega, \mathcal{A}, \mathsf{P})$, where Ω is the sample description space, $\mathcal{A}$ is a σ-algebra over Ω, and P is a probability measure on $\mathcal{A}$, is called a probability space.

The following theorem is essential for the probability calculus we discuss in the following.

Theorem A.8 *Let* P *be a probability measure on* $\mathcal{A}$.

1. $\mathsf{P}[\emptyset] = 0$

2. $\mathsf{P}[A \cup B] = \mathsf{P}[A] + \mathsf{P}[B]$ *for mutually exclusive* $A, B \in \mathcal{A}$

3. $\mathsf{P}[A \cup B] = \mathsf{P}[A] + \mathsf{P}[B] - \mathsf{P}[A \cap B]$ *for arbitrary* $A, B \in \mathcal{A}$

4. *if* $A \subseteq B$ *then* $\mathsf{P}[A] \leq \mathsf{P}[B]$

5. *if* $A \subseteq B$ *then* $\mathsf{P}[B \backslash A] = \mathsf{P}[B] - \mathsf{P}[A]$

6. $\mathsf{P}[\mathsf{C}A] = 1 - \mathsf{P}[A]$ *for every* $A \in \mathcal{A}$

Definition A.9 Let $(\Omega, \mathcal{A}, \mathsf{P})$ be a probability space as defined in A.7. Then a mapping $\mathsf{Z} : \Omega \to \mathbb{R}$ is called a random variable if and only if

$$\forall B \in \mathcal{B} : \quad \mathsf{Z}^{-1}(B) := \{\omega \in \Omega | \mathsf{Z}(\omega) \in B\} \in \mathcal{A} \tag{A.4}$$

It can be easily seen that P_Z is a probability measure on $\mathcal{B}$ with the definition $\mathsf{P}_\mathsf{Z}[B] := \mathsf{P}[\mathsf{Z}^{-1}(B)]$. This probability measure is called the distribution of Z.

P_Z is a set function and, therefore, hard to handle. It is, of course, desireable to have a simplies description of the distribution.

Definition A.10 Let Z be a random variable over the probability space $(\Omega, \mathcal{A}, \mathsf{P})$. Then the mapping

$$\begin{array}{rccl} \mathsf{F}_\mathsf{Z} : & \mathbb{R} & \longrightarrow & [0,1] \\ & x & \longmapsto & \mathsf{F}_\mathsf{Z}(x) := \mathsf{P}[\mathsf{Z}^{-1}((-\infty, x))] \end{array} \tag{A.5}$$

is called the distribution function of Z.

Definition A.11

1. If $\mathrm{Im}(\mathsf{Z})$ is a finite or countable set, the random variable Z is called discrete. If this is the case, the mapping

$$\begin{array}{rccl} \mathsf{f}_\mathsf{Z} : & \mathbb{R} & \longrightarrow & \mathbb{R}^+ \\ & x & \longmapsto & \mathsf{f}_\mathsf{Z}(x) := \mathsf{P}[\mathsf{Z}^{-1}(\{x\})] \end{array} \tag{A.6}$$

   is called the (probability) density function of Z.

2. If the probability function F_Z can be represented by

$$\mathsf{F}_\mathsf{Z}(x) = \int_{-\infty}^{x} \mathsf{f}_\mathsf{Z}(x) dx, \tag{A.7}$$

   where $\mathsf{f}_\mathsf{Z} : \mathbb{R} \to \mathbb{R}^+$ is a piecewise continuous function, the random variable Z is called continuous. Again the mapping f_Z is called probability density function of Z.

Definition A.12 Two arbitrary random variables X, Y over a probability space $(\Omega, \mathcal{A}, \mathsf{P})$ are called independent if and only if

$$\forall A, B \in \mathcal{B}: \quad \mathsf{P}[\mathsf{Z}^{-1}(A) \cap \mathsf{Z}^{-1}(B)] = \mathsf{P}[\mathsf{Z}^{-1}(A)] \cdot \mathsf{P}[\mathsf{Z}^{-1}(B)]. \tag{A.8}$$

Definition A.13

1. A discrete random variable is called integrable if and only if

   $$\sum_{\mathsf{f}_\mathsf{Z}(x)>0} |x| \cdot \mathsf{f}_\mathsf{Z}(x) < \infty.$$

   In this case

   $$\mathsf{E}[\mathsf{Z}] := \sum_{\mathsf{f}_\mathsf{Z}(x)>0} x \cdot \mathsf{f}_\mathsf{Z}(x) \tag{A.9}$$

   is called expectation of random variable Z.

2. A continuous random variable is called integrable if and only if

   $$\int_{-\infty}^{\infty} |x| \cdot \mathsf{f}_\mathsf{Z}(x)dx < \infty.$$

   In this case

   $$\mathsf{E}[\mathsf{Z}] := \int_{-\infty}^{\infty} x \cdot \mathsf{f}_\mathsf{Z}(x)dx \tag{A.10}$$

   is called expectation of random variable Z.

The following theorem provides some fundamental identities.

Theorem A.14 *Let X and Y be arbitrary random variables over a probability space $(\Omega, \mathcal{A}, \mathsf{P})$.*

1. *If X and Y are integrable, then also $\alpha\mathsf{X} + \beta\mathsf{Y}$ is integrable, where α and β are arbitrary real numbers. In addition, the following equation holds:*

   $$\mathsf{E}[\alpha\mathsf{X} + \beta\mathsf{Y}] = \alpha\mathsf{E}[\mathsf{X}] + \beta\mathsf{E}[\mathsf{Y}] \tag{A.11}$$

2. *If X and Y are independent and $\mathsf{X} \cdot \mathsf{Y}$ is integrable, the equation*

   $$\mathsf{E}[\mathsf{X} \cdot \mathsf{Y}] = \mathsf{E}[\mathsf{X}] \cdot \mathsf{E}[\mathsf{Y}] \tag{A.12}$$

   holds.

Bibliography

[Bagley, 1967] J. D. Bagley. *The Behavior of Adaptive Systems Which Employ Genetic and Correlative Algorithms.* PhD thesis, University of Michigan, Ann Arbor, 1967.

[Bauer *et al.*, 1993] P. Bauer, E. P. Klement, A. Leikermoser, and B. Moser. Approximation of real functions with rulebases. In *Proceedings of the Fifth IFSA World Congress*, pages 239–241. IFSA, 1993.

[Bauer *et al.*, 1995] P. Bauer, E. P. Klement, B. Moser, and A. Leikermoser. Modeling of control functions by fuzzy controllers. In H. T. Nguyen, M. Sugeno, R. Tong, and R. R. Yager, editors, *Theoretical Aspects of Fuzzy Control*, chapter 5, pages 91–116. John Wiley & Sons, Inc., 1995.

[Bauer, 1994] P. Bauer. Fitting of control functions by fuzzy methods with *mathematica.* Master's thesis, University of Linz, 1994.

[Belew and Booker, 1991] R. K. Belew and L. B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, California, 1991.

[Bonarini, 1993] A. Bonarini. ELF: learning incomplete fuzzy rule sets for an autonomous robot. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, volume I, pages 69–75, 1993.

[Buckley and Hayashi, 1994] J. J. Buckley and Y. Hayashi. Fuzzy genetic algorithm and applications. *Fuzzy Sets and Systems*, 61:129–136, 1994.

[Buckley, 1993] J. J. Buckley. Sugeno type controllers are universal controllers. *Fuzzy Sets and Systems*, 53:299–304, 1993.

[Butnariu and Klement, 1993] D. Butnariu and E. P. Klement. *Triangular Norm-Based Measures and Games with Fuzzy Coalitions.* Kluwer Academic Publishers, Dordrecht, 1993.

[Castro *et al.*, 1993] J. L. Castro, M. Delgado, and F. Herrera. A learning method of fuzzy reasoning by genetic algorithms. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, volume II, pages 804–809, 1993.

[Černy, 1985] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Opt. Theory Appl.*, 45:41–51, 1985.

[Chen and Chang, 1993] C. L. Chen and M. H. Chang. An enhanced genetic algorithm. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, volume II, pages 1105–1109, 1993.

[Darwin, 1991] C. R. Darwin. *On the Origin of Species by means of Natural Selection* and *The Descent of Man and Selection in Relation to Sex*, volume 49 of *Great Books of the Western World,* Editor in chief: M. J. Adler. Robert P. Gwinn, Chicago, Illinois, third edition, 1991. First edition John Murray, London, 1859.

[Daubechies, 1989] I. Daubechies. Orthonormal bases of wavelets with finite support-connection with discrete filters. In J. M. Combes, A. Grossmann, and P. Tchamitchian, editors, *Wavelets.* Springer, Berlin, Heidelberg, New York, 1989.

[Davis, 1987] L. Davis, editor. *Genetic Algorithms and and Simulated Annealing.* Research Notes in Artificial Intelligence. Pitman Publishing, London, 1987.

[Davis, 1991] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[Eshrag and Mamdani, 1979] F. Eshrag and E. H. Mamdani. A general approach to linguistic approximation. *International Journal of Man-Machine Studies*, 11:501–519, 1979.

[Feller, 1968] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I of *Wiley Series in Probability and Mathematical Statistics.* John Wiley & Sons, third revised edition, 1968.

[Feller, 1971] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume II of *Wiley Series in Probability and Mathematical Statistics.* John Wiley & Sons, second edition, 1971.

[Forrest, 1993] S. Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1993.

[Fukuda *et al.*, 1995] T. Fukuda, Y. Hasegawa, and K. Shimojima. Structure organization of hierarchical fuzzy model using by genetic algorithm.

In *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems*, volume I, pages 295–300. IEEE, 1995.

[Furuhashi *et al.*, 1995] T. Furuhashi, K. Nakaoka, and Y. Uchikawa. An efficient finding of fuzzy rules using a new approach to genetic based machine learning. In *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems*, volume II, pages 715–722, 1995.

[Gerhardt *et al.*, 1976] A. Gerhardt, J. Dircksen, and P. Höner. *Biologie für die Sekundarstufe I, Band I.* Bayerischer Schulbuch-Verlag, München, 1976. in German.

[Geyer-Schulz, 1995] A. Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness*. Physica Verlag, Heidelberg, 1995.

[Geyer-Schulz, 1996] A. Geyer-Schulz. The MIT beer distribution game revisited: Genetic machine learning and managerial behavior in a dynamic decision making experiment. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*. Physica Verlag, Heidelberg, 1996. to appear.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[Gonzalez *et al.*, 1993] A. Gonzalez, R. Perez, and J. L. Verdegay. Learning the structure of a fuzzy rule: A genetic approach. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, volume II, pages 814–819, 1993.

[Holland *et al.*, 1986] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* Computational Models of Cognition and Perception. The MIT Press, Cambridge, Massachusetts, 1986.

[Holland *et al.*, 1987] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. Classifier systems, q-morphisms, and induction. In L. Davis, editor, *Genetic Algorithms and and Simulated Annealing*, Research Notes in Artificial Intelligence, pages 116–128. Pitman Publishing, London, 1987.

[Holland, 1986] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalsky, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, volume II*, pages 593–623. Morgan Kaufmann, Los Altos, 1986.

[Holland, 1992] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, Massachusetts, first MIT Press edition, 1992. First edition: University of Michigan Press, 1975.

[Ishibuchi *et al.*, 1993] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka. Genetic operations for rule selection in fuzzy classification systems. In *Proceedings of the Fifth IFSA World Congress*, volume I, pages 15–18. IFSA, 1993.

[Kacprzyk, 1995] J. Kacprzyk. Multistage control of a fuzzy system using a genetic algorithm. In *Proceedings of the Fourth IEEE International Conference*, volume III, pages 1083–1088. IEEE, 1995.

[Karr, 1991] C. L. Karr. Design of an adaptive fuzzy logic controller using a genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1991.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Klement and Slany, 1993] E. P. Klement and W. Slany, editors. *Fuzzy Logic in Artificial Intelligence*, volume 695 of *Lecture Notes in Artificial Intelligence.* Springer, Berlin, Heidelberg, June 1993.

[Kosko, 1992] B. Kosko. Fuzzy systems as universal approximators. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1153–1162. IEEE, 1992.

[Koza, 1992] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[Kruse *et al.*, 1994] R. Kruse, J. Gebhardt, and F. Klawonn. *Foundations of Fuzzy Systems.* John Wiley & Sons, first edition, 1994.

[van Laarhoven and Aarts, 1987] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, Dordrecht/Boston/London, 1987.

[Lee and Takagi, 1993a] M. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1993.

[Lee and Takagi, 1993b] M. Lee and H. Takagi. Embedding apriori knowledge into an integrated fuzzy system design method based on genetic algorithms. In *Proceedings of the Fifth IFSA World Congress*, pages Vol. II, 1293–1296. IFSA, 1993.

[Lee and Takagi, 1993c] M. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, volume I, pages 612–617. IEEE, 1993.

[Lin and Chen, 1995] S. C. Lin and Y. Y. Chen. A GA-based fuzzy controller with sliding mode. In *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems*, volume III, pages 1103–1110. IEEE, 1995.

[Linder, 1979] A. Linder. *Biologie, Teil 2.* Verlag Gustav Swoboda & Bruder, Wien, 1979. in German.

[Magdalena and Monasterio, 1995] L. Magdalena and F. Monasterio. Evolutionary-based learning applied to fuzzy controllers. In *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems*, volume III, pages 1111–1118. IEEE, 1995.

[Mamdani and Assilian, 1975] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis of fuzzy controllers. *International Journal of Man Machine Studies*, 7:1–13, 1975.

[Mamdani, 1977] E. H. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic systems. *IEEE Transactions on Computation*, 26:1182–1191, 1977.

[Metropolis *et al.*, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chem. Physics*, 21:1087–1092, 1953.

[Michalsky *et al.*, 1986] R. S. Michalsky, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning, volume II.* Morgan Kaufmann, Los Altos, 1986.

[Mitsubuchi *et al.*, 1993] K. Mitsubuchi, S. Isaka, and Z. Y. Zhao. A fuzzy rule generation system. In *Proceedings of the Fifth IFSA World Congress*, volume I, pages 11–14. IFSA, 1993.

[Moser, 1996] B. Moser. *A New Approach for Representing Control Surfaces by Fuzzy Rule Bases.* PhD thesis, University of Linz, 1996.

[Neunzert and Wetton, 1987] H. Neunzert and B. Wetton. Pattern recognition using measure space metrics. Technical Report 28, Universität Kaiserslautern, Fachbereich Mathematik, November 1987.

[Nguyen *et al.*, 1995] H. T. Nguyen, M. Sugeno, R. Tong, and R. R. Yager, editors. *Theoretical Aspects of Fuzzy Control.* John Wiley & Sons, 1995.

[Novák, 1989] V. Novák. *Fuzzy Sets and Their Applications.* Adam-Hilger, Bristol, 1989.

[Otten and van Ginneken, 1989] R. H. J. M. Otten and L. P. P. P. van Ginneken. *The Annealing Algorithm.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1989.

[Parzen, 1960] E. Parzen. *Modern Probability Theory and Its Application.* John Wiley & Sons, Inc., 1960.

[Rechenberg, 1973] I. Rechenberg. *Evolutionsstrategie*, volume 15 of *Problemata.* Friedrich Frommann Verlag (Günther Holzboog KG), Stuttgart, 1973. in German.

[Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing — Exploration in the Microstructures of Cognition, Volume I: Foundations.* The MIT Press, Cambridge, Massachusetts, 1986.

[Ruspini, 1969] E. H. Ruspini. A new approach to clustering. *Information Control*, 15:22–32, 1969.

[Schwefel and Männer, 1991] H.-P. Schwefel and R. Männer, editors. *Parallel Problem Solving from Nature.* Springer, Berlin, 1991.

[Shimojima *et al.*, 1995] K. Shimojima, T. Fukuda, and Y. Hasegawa. Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *Fuzzy Sets and Systems*, 71(3):295–309, 1995.

[Stark, 1990] H. G. Stark. Multiscale analysis, wavelets, and texture quality. Technical Report 41, Universität Kaiserslautern, Fachbereich Mathematik, January 1990.

[Sugeno, 1985] M. Sugeno. An introductory survey of fuzzy control. *Information Sciences*, 36:59–93, 1985.

[Surmann *et al.*, 1993] H. Surmann, A. Kanstein, and K. Goser. Self-organizing and genetic algorithms for an automatic design of fuzzy control and decision systems. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, volume II, pages 1097–1104, 1993.

[Takagi and Lee, 1993] H. Takagi and M. Lee. Neural networks and genetic algorithms to auto design of fuzzy systems. In E. P. Klement and W. Slany, editors, *Lecture Notes in Artificial Intelligence*, volume 695, pages 68–79. Springer, 1993.

[Takagi and Sugeno, 1985] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Sytems, Man, and Cybernetics*, 15(1):116–132, 1985.

[Thrift, 1991] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.

[Valenzuela-Rendón, 1991a] M. Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 346–353, San Mateo, California, 1991. Morgan Kaufmann.

[Valenzuela-Rendón, 1991b] M. Valenzuela-Rendón. The fuzzy classifier system: Motivations and first results. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 330–334. Springer, Berlin, 1991.

[Wang, 1992] L. X. Wang. Fuzzy systems are universal approximators. In *Proceedings of the First IEEE International Conference on Fuzzy Systems*, pages 1163–1169. IEEE, 1992.

[Weyl, 1916] H. Weyl. Über die Gleichverteilung von Zahlen mod. Eins. *Math. Ann.*, 77:313–352, 1916. in German.

[Yager *et al.*, 1987] R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, editors. *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh*. John Wiley & Sons, Inc., New York, 1987.

[Yager, 1980] R. R. Yager. Aspects of possibilistic uncertainty. *International Journal of Man-Machine Studies*, 12:283–298, 1980.

[Yager, 1983] R. R. Yager. An introduction to applications of possibility theory. *Human Systems Management*, 3:246–269, 1983.

[Yubazaki *et al.*, 1995] N. Yubazaki, M. Otani, T. Ashida, and K. Hirota. Dynamic fuzzy control method and its application to positioning of induction motor. In *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems*, volume III, pages 1095–1102. IEEE, 1995.

[Zadeh, 1965] L. A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[Zadeh, 1987a] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning I. In R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, editors, *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh*, pages 219–269. John Wiley & Sons, Inc., New York, 1987.

[Zadeh, 1987b] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning II. In R. R. Yager, S. Ovchinnikov,

R. M. Tong, and H. T. Nguyen, editors, *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh*, pages 271–327. John Wiley & Sons, Inc., New York, 1987.

[Zadeh, 1993] L. A. Zadeh. The role of fuzzy logic and soft computing in the conception and design of intelligent systems. In E. P. Klement and W. Slany, editors, *Lecture Notes in Artificial Intelligence*, volume 695, page 1. Springer, 1993.

[Zurada, 1992] J. M. Zurada. *Introduction to Artificial Neural Networks.* West Publishing, St. Paul, 1992.

Index

Note: Underlined page numbers refer to especially important occurrences of the entry, such as exact mathematical definitions.

Me About Myself — A Brief Curriculum Vitae

I was born on the 24th of April 1972 at Braunau hospital as first of, at least until now, two children of Josef and Elisabeth Bodenhofer, a teacher and a florist, respectively. Subsequently, I grew up in Altheim, a village in the west of Upper Austria with approximately five thousand inhabitans.

In September 1978 I entered primary school in Altheim which I completed in July 1982. In September of the same year, I entered Braunau grammar school, where I passed my final exam ("Matura") in June 1990 with distinction.

In October 1990 I moved to Linz and began to study "Technical Mathematics" at the Johannes Kepler University there. In the winter semester 1992/93, when I attended a lecture on fuzzy control by Prof. Erich Peter Klement, I got in contact with fuzzy logic the very first time. Impressed by the simplicity and effectivity of this new technique I attended a seminar in the following summer semester where I developed a fuzzy controller for a sand paper drying machine together with two colleagues.

At the beginning of the 1994 summer semester Prof. Klement asked me to join a project study concerning with image segmentation, some results of which are discussed in chapter 4. After a successful completion, this project study was extended to a large project on which I worked between August 1994 and March 1995. It dealt with the development of a complete inspection system for a silk-screen printing process. This work included the development of new and the application of known mathematical algorithms for image processing, the design of a fuzzy expert system for the quality decision as well as the implementation of an X-Windows based user interface and routines for the communication with a camera and a serial port. Most of the work was done by myself, Markus Mittendorfer, and Stephan Nouak,, two collegues at the *FLLL*, a member of which I had become in the meantime. After a three months long stay at our partner company near Salzburg, where I finished the project, I returned to Linz and started to work on this thesis.