FUZZY SYSTEMS AND GENETIC ALGORITHMS

Béla Lantos

Budapest University of Technology and Economics
Department of Control Engineering and Information Technology

Lecture Notes

Budapest, 2001

Preface

Intelligent systems need the application of both old type (mathematics based) and new type (fuzzy, neural and genetic algorithm based) computational methods. Fuzzy systems (FS), neural networks (NN) and genetic algorithms (GA) are relatively new methods in computational intelligence.

Fuzzy systems are defined by membership functions of the linguistic variables, the relations describing the system behavior in fuzzy sense, the fuzzification rule, the inference rule and the defuzzification rule. Since the relations normally use standardized variables, input and output normalizing gains belong also to the system parameters. If the consequent parts of the relations are deterministic functions the fuzzy system is of Takagi-Sugeno-Kong (TSK) type. Especially if the deterministic functions are linear (or constant linear) then the FS is of Sugeno type. It is known that every continuous function over a compact interval can be approximated by Sugeno type FS with constant output deterministic functions. For increasing number of variables it is impossible to find a priori the relations of the FS, hence adaptive tuning is necessary. Sugeno type FS can be rewritten to a feedforward NN, thus the gradient method of optimization in the form of backpropagation (BP) can also be used.

A feedforward NN consists of one input, some hidden and one output layer. A neuron is characterized by the weights for the input signals and the transition function mapping the net input (weighted sum of the input signals) into the output of the neuron. Several type of transition functions can be used (sigmoid, hard limits, linear etc.). The feedforward NN has also function approximation properties.

Both FS and feedforward NN can be converted into feedforward adaptive networks (AN). AN's are similar to NN's with the only difference that the weights of the NN are removed into the arguments of the functions placed in the nodes of the NN. If the node does not contain any parameters it is a fixed node, otherwise it is an adaptive node. If some of the parameters appear linear in the functions then linear parameter estimation can be applied for them, while gradient method etc. for the remaining parameters in the actual step of the tuning (hybrid optimization).

Since gradient, conjugate gradient and quasi-Newton methods are usually convergent to local optimum only, methods are needed for finding the global optimum. Since genetic and evolutionary algorithms are stochastic optimization methods they have the chance to find the global optimum (minimum) of a scalar-valued function. The input variables of the objective function can be coded in standard binary or Gray-code (binary GA) or remain real uncoded (real GA). In order to improve the stochastic property the value of the objective function will be mapped into a bounded domain by using a fitness function (linear ranking, nonlinear ranking). The population consists of individuals (input vectors). Parents will be selected from the population using roulette-wheel selection, stochastic universal sampling etc. Between the pairs of parents recombination (crossover) will be performed and with small probability mutation is applied. The children will be replaced into the population based on the fitness of the new individuals but saving the best ones in the old population (elitist strategy). In order to improve the stochastic properties multipopulation genetic algorithms (MPGA) can also be used where the

population consists of subpopulations. In this case migration between the subpopulations is possible.

In many cases if the optimal FS, NN or AN has to be found for given input-output samples, the best strategy is to start with optimization using GA. Unfortunately by using genetic operators of the GA, enormly large number of steps are needed to reach the global optimum because stochastic optimization is time consuming. Hence the GA will often be used to come near to the global optimum, and the optimization will be continued using gradient (BP), conjugate gradient, quasi-Newton optimum seeking methods or linear parameter estimation (LS), nonlinear LS or extended Kalman filter techniques.

Intelligent systems have the fundamental function of sensing, perception, knowledge acquisition, learning, inference, decision making and acting abilities. Soft computing (FS, NN, AN, GA) methods can help to realize these functions. They can deal as alternative methods in system modeling, optimal and robust control design, nonlinear and adaptive control too. They can support the design and realization of autonomous (wheeled or legged) mobile robots, service robots, cooperating robots, robotized production systems, aircraft and satellite control systems, and the modeling of complex (technical and non-technical) processes.

The book is based on the lectures in Fuzzy Systems and Genetic Algorithms which have been held by the author at the Faculty of Electrical Engineering and Informatics of the Budapest University of Technology and Economics (former Technical University of Budapest) since 1997.

The first two chapters of the book summarize the fundamentals of fuzzy systems and (single objective) genetic algorithms. Chapter 3 deals with optimum seeking methods and linear and nonlinear parameter estimation. Chapter 4 gives methods for automatic rule base design of fuzzy systems including clustering, stochastic embedding and evolutionary techniques. Chapter 5 deals with neuro-fuzzy systems especially the ANFIS (adaptive neuro-fuzzy inference system) approach. Chapter 6 deals with adaptive fuzzy control of single variable (SISO) and multivariable (MIMO) systems. Chapter 7 concentrates on optimal fuzzy modeling based on input-output data over a rectangular grid. Chapter 8 deals with multiobjective genetic algorithms. References contain the basic literature used in the book.

The book can be applied well in the engineering and informatics education at universities, and can be suggested for PhD students and people in education and research. The full content is planned for one semester curriculum (4 hours times 14 weeks) in Fuzzy Systems and Genetic Algorithms. Some parts of the book (for example fundamentals of fuzzy systems, adaptive neuro-fuzzy systems and adaptive fuzzy control) can be used separately in a Control Theory curriculum too.

Acknowledgements: This work was supported by the Tempus Phare Joint European Project S_JEP 12555-97, by the Hungarian National Research Fonds OTKA T 029072 and FKFP 0417/1997, and by the Control Research Group of the Hungarian Academy of Sciences, which are gratefully acknowledged.

Budapest, September 2001.

Professor Dr Béla Lantos

Contents

1. FUNDAMENTALS OF FUZZY SYSTEMS

1.1 Fuzzy sets and membership functions

In classical set theory it is usual to talk about a subset A of a universe of discourse X, in which an arbitrary element $x \in X$ is either an element of A, or definitely not an element of A, see *Fig. 1.1*. In the latter case x is an element of the complement of A (A^c).

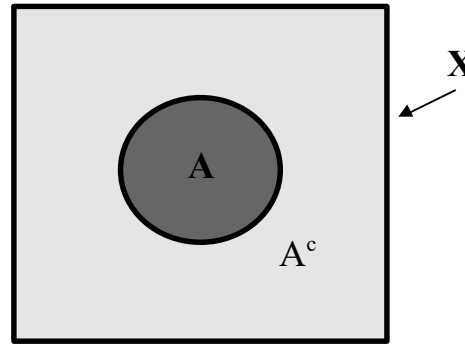


Fig. 1.1 Classical set concept

We can assign to set A its characteristic function $k_A : X \to \{0,1\}$, which can take only 0 or 1 (crisp) numerical values:

$$k_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases}$$

It is clear that $k_{A^c} = 1 - k_A$. Since the characteristic function k_A unambiguously distinguishes the elements of A from the other elements of X $(A^c = X \setminus A)$, thus A can be identified with the pair (X, k_A):

$$A \leftrightarrow (X, k_A)$$

The human reasoning often operates with uncertain linguistic expressions which, precisely seen, different people can interpret in a subjective manner. For instance, it

is frequently told about a temperature J that it is cold, cool, comfortable, warm or hot, and not everybody mean precisely the same. For example, if the temperature falls into the interval [-15°C, +35°C] in a given geographic region then different people imagine different temperature ranges to the word "cool" and they do not think each temperature value of this interval crisply "cool". If in automated systems inferences should be made by using such uncertain ideas and online human support (intelligence, experience, etc.) is not present, then it is useful to make definable the notion of uncertainty and work out the rules of operations which manipulates with uncertain definitions so that automated systems act similarly like human. However, it appears to be natural that the definability of the notion of uncertainty and the capability of choosing from the offered rules to be evolved bring some subjectivity to the attributes of the automatic system to be developed. For the description of uncertain linguistic notions, Zadeh had proposed the notion of fuzzy set [1], which have been used in several automated systems since then [2],[4],[5],6],[9],[44],[45].

We can attach the temperature interval [-5°C, +5°C] to the "cool" linguistic variable. In the center of this interval it is surely cool (possibility 1), and the possibility of to be cool is gradually decreases toward the boundaries of the interval (from 1 gradually converging value toward 0 if the temperature approaches from the center toward the boundary of the interval). It is certainly not cool outside of the interval (possibility 0). Thus a fuzzy set can be attached to the "cool" linguistic variable, which is defined by the function $m_{\text{cool}} : [-15^{\circ}C, -35^{\circ}C] \rightarrow [0,1]$, which is called the membership function of the fuzzy set.

Generally, if X is the universe of discourse and A is a fuzzy set on X, then the degree to which an element $x \in X$ belongs to A is defined by $m_A(x) \in [0,1]$. It is apparent that the fuzzy membership function m_A replaces the crisp characteristic function k_A. The fuzzy set A can be identified with the pair (X, m_A):

$$\text{Fuzzy set } A \leftrightarrow (X, m_A)$$

The fundamental knowledge representation unit in approximate reasoning is the notion of a linguistic variable. By a linguistic variable we mean a variable whose values are words or sentences in a natural or artificial language. The values of the linguistic variable are linguistic rather than numerical (Zadeh, [3]). It is usual in approximate reasoning to have the following *5*-tuple associated with the notion of a linguistic variable [5]:

$$(x, LX, \widetilde{L}\widetilde{X}, X, M_x)$$

Here x is the symbolic name of the linguistic variable. $LX = \{X^1, X^2, \ldots, X^{n_x}\}$ is the set of the linguistic values that the linguistic variable x can take (term-set or reference-set of x). X is the actual physical domain (universe of discourse) over which the linguistic variable x takes its quantitative (crisp) values. $\widetilde{LX} = \{\widetilde{X}^1, \widetilde{X}^2, \ldots, \widetilde{X}^{n_x}\}$ is the fuzzy characterisation of the linguistic values over X. The semantic function $M_x : LX \to \widetilde{LX}$ gives a meaning (interpretation) of a linguistic value in terms of quantitative elements of X.

In the above example we have:

$x = J$,

$LX = \{X^{\text{cold}}, X^{\text{cool}}, X^{\text{comfortable}}, X^{\text{warm}}, X^{\text{hot}}\}$:={cold,cool,comfortable,warm,hot},

$\widetilde{LX} = \{\widetilde{X}^{\text{cold}}, \widetilde{X}^{\text{cool}}, \widetilde{X}^{\text{comfortable}}, \widetilde{X}^{\text{warm}}, \widetilde{X}^{\text{hot}}\}$,

X =[-15°C,+35°C],

$M_x : X^{\text{cold}} \mapsto \widetilde{X}^{\text{cold}}, \cdots, M_x : X^{\text{hot}} \mapsto \widetilde{X}^{\text{hot}}$.

We shall introduce the following simplified notion:

$$(x, LX, \widetilde{LX}, X, M_x) \leftrightarrow (x, \{X^1 = (X, m_{X^1}), \cdots, X^{n_x} = (X, m_{X^{n_x}})\}.$$

It means that in the following we do not distinguish between the linguistic value and the assigned fuzzy set, for example $\text{cool} \leftrightarrow (X, m_{\text{cool}})$.

There are several conventions to define fuzzy sets. In many applications we discretize (quantize) the set X, thus $X = \{x_1, \ldots, x_n\}$ and $m_A : \{x_1, \ldots, x_n\} \to [0,1]$ is the membership function of the fuzzy set A. In this case the fuzzy set A can be defined by a symbolic sum, which contains the symbolic pair $m_A(x_i)/x_i$ (and not the ratio):

$$A = \sum_{i=1}^{n} m_A(x_i)/x_i$$

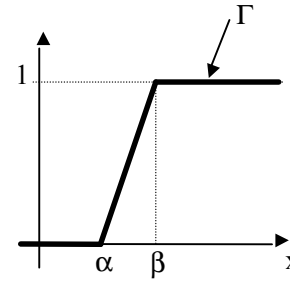
Similarly, if X is a continuous (not discrete, but analogous) set, then $m_A : X \to [0,1]$ and A is a fuzzy set, therefore $A = (X, m_A)$ can be specified by a symbolic integral, which contains the symbolic pair $m_A(x)/x$:

$$A = \int_{x \in X} m_A(x)/x$$

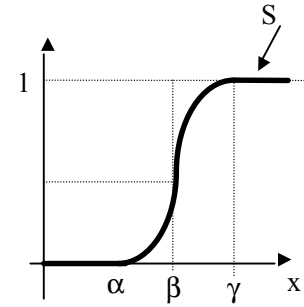
Since it is not a real integral, but only a notation, thus it is not permitted to use dx.

In discrete case the membership function can be defined with a table or a symbolic sum. In continuous case, if x is a scalar variable, then the typical form of $m_A(x)$ can be for instance a Gaussian distribution curve, or a piecewise linear function. In the latter case, depending on the shape of the curve we can talk about Γ,S,L,Λ,Π curves, where the Latin or Greek letters describe the shape of the curve, see *Fig. 1.2.*

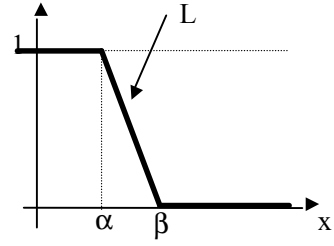
$$G(x,a,b)=\begin{cases}0, & if\ x<a\\ \dfrac{x-a}{b-a}, & if\ x\in[a,b]\\ 1, & if\ x>b\end{cases}$$



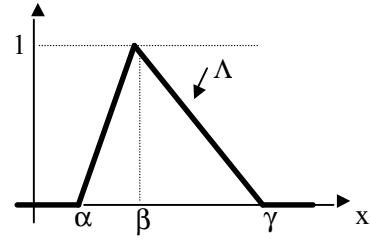
$$S(x,a,b=\frac{a+g}{2},g)=\begin{cases}0, & \text{if } x<a\\ 2\left(\dfrac{x-a}{g-a}\right)^2, & \text{if } x\in[a,b]\\ 1-2\left(\dfrac{x-g}{g-a}\right)^2, & \text{if } x\in[b,g]\\ 1, & \text{if } x>g\end{cases}$$



$$L(x,a,b)=1-G(x,a,b)=\begin{cases}1, & if\ x<a\\ \dfrac{b-x}{b-a} & if\ x\in[a,b]\\ 0, & if\ x>b\end{cases}$$



$$L(x,a,b,g)=\begin{cases}0, & \text{if } x<a\\ \dfrac{x-a}{b-a}, & \text{if } x\in[a,b]\\ \dfrac{x-g}{b-g}, & \text{if } x\in[b,g]\\ 0, & \text{if } x>g\end{cases}$$



$$P(x,a,b,g,d)=\begin{cases}0, & \text{if } x<a\\ \dfrac{x-a}{b-a}, & \text{if } x\in[a,b]\\ 1, & \text{if } x\in[b,g]\\ \dfrac{d-x}{d-g}, & \text{if } x\in[g,d]\\ 0, & \text{if } x>d\end{cases}$$

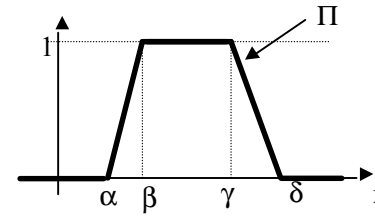


Fig. 1.2 Typical membership functions

We often normalize the physical variable to a region and distinguish the fuzzy sets (linguistic values) by the sign (P=positive, N=negative) and magnitude (B=big, M=medium, S=small, Z=zero), for instance we distinguish 7 cases on the [-6,+6] interval (from the left to the right NB, NM, NS, Z, PS, PM, PB), see *Fig. 1.3*.

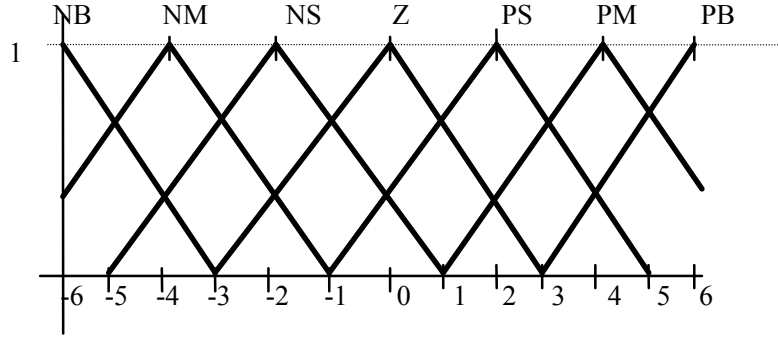


Fig. 1.3 Standard region with 7 levels (equipartition)

The number of linguistic variables can be increased by further differentiation by magnitude (VL=very large, L=large, JL=just large, M=medium, JM=just medium, S=small, Z=zero), see *Fig. 1.4*. Instead of equipartition, a finer partition (higher level of sensitivity) can be used in the neighborhood of zero, for example by further subdivisions (S=small, VS=very small), see *Fig. 1.5-1.7*.

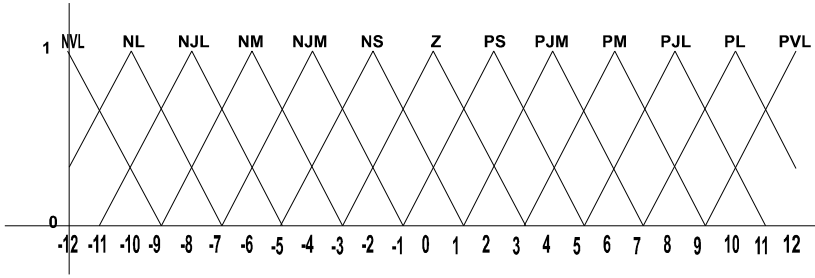


Fig. 1.4 Standard region with 13 levels (equipartition)

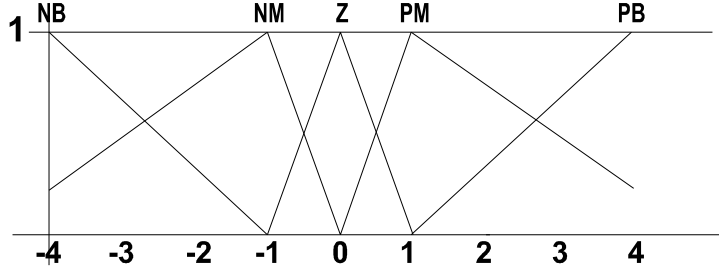


Fig. 1.5 Non-uniform partition with 5 levels

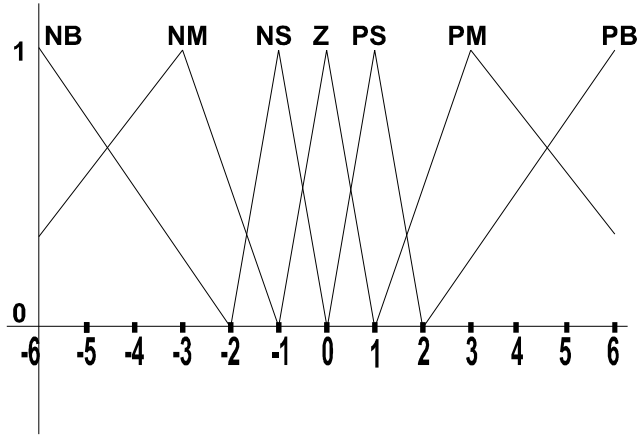


Fig. 1.6 Non-uniform partition with 7 levels

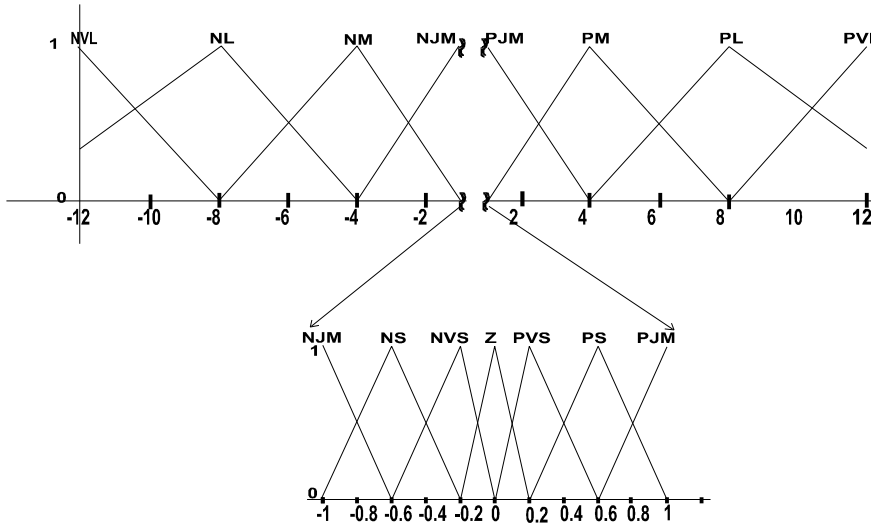


Fig. 1.7 Non-uniform partition with 13 levels

Gaussian and bell-type membership functions are very popular to describe a linguistic value (fuzzy set) in continuous way.

Gaussian membership function:

$m(x) = e^{-\frac{1}{2}\left(\frac{x-\bar{x}}{s}\right)^2}$, $\bar{x}$ is the center and s is the width.

Generalized bell membership function:

$$m(x) = \frac{1}{1+\left|\frac{x-\bar{x}}{a}\right|^{2b}},$$ $\bar{x}$ is the center and a, b define the width.

1.2 Elementary fuzzy operations and norms

1.2.1 Fuzzy union, intersection and complement

In the area of fuzzy sets we can define set operations, such as fuzzy union ($\cup$), fuzzy intersection ($\cap$) and fuzzy complement. In this regard, knowing the m_A and

m_B membership functions, computing rules are needed for the membership functions $m_{A\cup B}$, $m_{A\cap B}$ and m_{A^c}. If the values of $m_A(x)$ and $m_B(x)$ are known, i.e. the possibility that x belongs to A or B respectively, then it is required that the possibility is not decreased in case of $m_{A\cup B}(x)$, it is not increased in case of $m_{A\cap B}(x)$ and it is $m_{A^c}(x)=1-m_A(x)$ (or similar) in case of complement set. Therefore, as a general rule, we should derive from the 'norms' of m_A and m_B the 'norms' $m_{A\cup B}$, $m_{A\cap B}$ and m_{A^c}, respectively. For this purpose there are several rules used with fuzzy sets, for instance the *S*-norms, *T*-norms and *c*-norms for the case of union, intersection and complement, respectively:

$$m_{A\cup B}(x)=S(m_A(x),m_B(x))=m_A(x)\vee m_B(x)$$

$$m_{A\cap B}(x)=T(m_A(x),m_B(x))=m_A(x)\wedge m_B(x)$$

$$m_{A^c}(x)=c(m_A(x))$$

These norms should satisfy some axioms.

1.2.2 *T*-, *S*- and *c*-norms

The *T*, *S* and *c*-norms are used for computation of membership functions of fuzzy intersection, union and complement respectively [5],[9].

T-norm and its properties

The *T*-norm should satisfy the following axioms:

$$T:[0,1]\times[0,1]\to[0,1],$$
$$T(a,b):=a\wedge b$$

i) $a\wedge b=b\wedge a$
ii) $(a\wedge b)\wedge c=a\wedge(b\wedge c)$
iii) $a\le c$ and $b\le d\Rightarrow a\wedge b\le c\wedge d$
iv) $a\wedge 1=a$

It follows from the properties that $0\wedge a=0$, because according to iii) and iv) $0\le 0$ and $a\le 1\Rightarrow 0\wedge a\le 0\wedge 1=0\Rightarrow 0\wedge a=0$.

The *T*-norm is Archimedean, if $a\wedge a<a$ for $\forall a\in(0,1)$.

Examples for *T*-norms:

$a\wedge b=\min\{a,b\}$ (minimum norm)

$a \wedge b = a \cdot b$ (algebraic product or dot norm)
$a \wedge b = \max\{0, a + b - 1\}$ (bounded product)

$$T_p(a,b) = 1 - [(1-a)^p + (1-b)^p - (1-a)^p (1-b)^p]^{1/p}$$

$$T_1(a,b) = ab$$

$$T_\infty(a,b) = \lim_{p\to\infty} T_p(a,b) = \min\{a,b\}$$

$$T_0(a,b) = \lim_{p\to 0} T_p(a,b) = T_w(a,b) := \begin{cases} a, \text{ if } b = 1 \\ b, \text{ if } a = 1 \\ 0, \text{ otherwise} \end{cases} \quad \text{(drastic product)}$$

Hamacher's *T*-norms:

$$H_l(a,b) = \frac{ab}{l + (1-l)(a+b-ab)}, \; l > 0$$

$$H_1(a,b) = ab$$

$$H_0(a,b) = \frac{ab}{a+b-ab}$$

$$H_\infty(a,b) = T_W(a,b)$$

Frank's *T*-norm:

$$F_s(a,b) = \log_s\left(1 + \frac{(s^a - 1)(s^b - 1)}{s-1}\right), \; s > 0$$

Yager's *T*-norm :

$$Y_q(a,b) = 1 - \min\left\{1, \left[(1-a)^q + (1-b)^q\right]^{1/q}\right\}, \; q \geq 0$$

Dubois-Prade's *T*-norms:

$$s_a(a,b) = \frac{ab}{\max(a,b,a)}$$

$$s_0 = \min(a,b)$$

$$s_1 = ab$$

$$a \in (0,1) \Rightarrow s_a(a,b) := \begin{cases} ab/a, \text{ if } a,b < a \\ \min(a,b), \text{ otherwise} \end{cases}$$

<u>*S*-norm (*T*-conorm) and its properties:</u>
The *S*-norm should satisfy the following axioms:

$$S : [0,1]\times[0,1] \to [0,1]$$
$$S(a,b) := a \vee b$$

i) $a \vee b = b \vee a$

ii) $(a \vee b) \vee c = a \vee (b \vee c)$

iii) $a \le c$ and $b \le d \Rightarrow a \vee b \le c \vee d$

iv) $a \vee 0 = a$

It follows from properties iii) and iv), that $a \vee 1 = 1$. Really, because $1 \le 1$ and $0 \le 1$ $\Rightarrow 1 = 1 \vee 0 \le 1 \vee 1 \in [0,1] \Rightarrow 1 \vee 1 = 1$, consequently it follows from $0 \le a$ and $1 \le 1$ that $0 \vee 1 = 1 \vee 0 = 1 \le a \vee 1 \in [0,1]$, thus $a \vee 1 = 1$.

Examples for *S*-norms:

$$a \vee b = \max\{a, b\}$$

Sugeno's *S*-norms:

$$S_l(a,b) = \min\{1, a + b + lab\}, \quad l \ge -1$$

$$S_0(a,b) = \min\{1, a + b\} \text{ (bounded sum)}$$

$$S_{-1}(a,b) = a + b - ab \text{ (algebraic sum)}$$

$$S_w(a,b) = \begin{cases} a, \text{ if } b = 0 \\ b, \text{ if } a = 0 \\ 1, \text{ otherwise} \end{cases} \text{ (drastic sum)}$$

c-norm and its properties:

The *c*-norm should satisfy the following axioms:

$$c : [0,1] \to [0,1]$$

i) $c(0) = 1$

ii) $a \le b \Rightarrow c(a) \ge c(b)$

iii) $c(c(a)) = a$

It is clear that because of i) and iii), $0 = c(c(0)) = c(1)$.

Examples for *c*-norms:

$$c(a) = 1 - a$$

Sugeno's *c*-norm:

$$c_s(a) = \frac{1-a}{1+sa}, \quad s > -1$$

Yager's *c*-norm:

$$c_w(a) = (1 - a^w)^{1/w}, \quad w > 0$$

Choosing appropriate *T*- and *S*-norms, we can add optimism and pessimism with various degrees to the fuzzy set operations intersection and union, because in case of the above $T(a,b)$ and $S(a,b)$ the following limits are valid:

$$T_w(a,b) \le T(a,b) \le \min\{a,b\}$$
$$\max\{a,b\} \le S(a,b) \le S_w(a,b)$$
pessimist optimist

1.3 Fuzzy set operations in Cartesian product spaces

After specifying fuzzy union, intersection and complement, we will examine the concept of direct product, relation, projection, cylindrical extension and composition of fuzzy sets, which have important role in applications.

1.3.1 Fuzzy direct product

Let X_i denote the base set, let A_i denote a fuzzy set with membership function $m_{A_i}(x_i)$, where $A_i = (X_i, m_{A_i})$, $i = 1,2,\ldots,n$. Since $x = (x_1, \ldots x_n)$ contains $x_1,\ldots,x_n$ simultaneously, the membership function of the fuzzy direct product $A_1 \times A_2 \times \cdots \times A_n$ is:

$$m_{A_1\times\cdots\times A_n}(x_1,\ldots,x_n) = m_{A_1}(x_1) \wedge \cdots \wedge m_{A_n}(x_n)$$
$$A_1 \times \cdots \times A_n \text{ fuzzy direct product} \leftrightarrow (X_1 \times \cdots \times X_n, m_{A_1\times\cdots A_n})$$

Here $\wedge$ is any appropriate *T*-norm, for example $\wedge = \min$ etc.

1.3.2 Fuzzy relation

In the classical meaning, the binary relation $x\,R\,y$ (for instance $x \equiv y$ or $x \le y$) specifies that some elements $(x,y) \in X \times X$ of the set are in relation R with each other, while the relation is not fulfilled by other $(x,y) \in X \times X$ elements. According to this, the binary relation R is a special $R \subset X \times X$ subset of the $X \times X$ direct product:

$$x\,R\,y \leftrightarrow (x,y) \in R \subset (X \times X).$$

Among binary relations, the following properties are typical:

reflective	xRy
symmetric	$xRy \Rightarrow yRx$
antisymmetric	xRy and $yRx \Rightarrow x = y$ (identical)
transitive	xRy *and* $yRz \Rightarrow xRz$

Especially, the equivalence relation $\equiv$ is reflective, symmetric and transitive, while the partial ordering relation $\leq$ is reflective, antisymmetric and transitive. Usually, the classical binary relation can be generalized for more than two variables, such as $R \subset X_1 \times \cdots \times X_n$. On the analogy of the above, the fuzzy relation R can be defined by the membership function on the variables $x_1, \ldots, x_n$:

$$m_R : X_1 \times \cdots \times X_n \to [0,1]$$
$$R \text{ fuzzy relation} \leftrightarrow (X_1 \times \cdots \times X_n, m_R)$$

Therefore, we have to define a fuzzy relation by the specification of the related membership function $m_R(x_1, \ldots, x_n)$, which gives the possibility of $(x_1, \ldots, x_n) \in R$, i.e. relation R holds among them in fuzzy sense. Since the fuzzy system realizes an action according to certain rules when certain conditions are fulfilled, in the algorithmization of this process the fuzzy relations will have important role.

After having defined certain fuzzy relations as special subsets of the product space $X_1 \times \cdots \times X_n$, then the set operation defined on fuzzy sets can be applied on them. However, since the set operations require the same base-set (universe of discourse) hence we have to pay particular attention to each relation has the same $X_1 \times \cdots \times X_n$ product set as its domain. If this condition is not satisfied then the relations should be extended to a common product set.

If R_1 and R_2 are fuzzy relations on the base set $X_1 \times \cdots \times X_n$, then the membership functions of their fuzzy union and intersection are:

$$m_{R_1 \cup R_2}(x_1, \ldots, x_n) = m_{R_1}(x_1, \ldots, x_n) \vee m_{R_2}(x_1, \ldots, x_n)$$
$$m_{R_1 \cap R_2}(x_1, \ldots, x_n) = m_{R_1}(x_1, \ldots, x_n) \wedge m_{R_2}(x_1, \ldots, x_n)$$
$$R_1 \cup R_2 = (X_1 \times \cdots \times X_n, m_{R_1 \cup R_2})$$
$$R_1 \cap R_2 = (X_1 \times \cdots \times X_n, m_{R_1 \cap R_2})$$

where $\vee$ and $\wedge$ are appropriate *S*-norms and T-norms respectively, for example $\vee = \max$ and $\wedge = \min$.

1.3.3 Cylindrical extension

If the domains of two relations are not the same product space, then each of them can be extended to a common product set particularly. Consider one of them. Let R be a fuzzy relation with arguments on the product space $X_{i_1} \times X_{i_2} \times \cdots \times X_{i_r}$, which we want to extend to the product space $X_1 \times \cdots \times X_n$, where $\{i_1, \ldots, i_r\} \subset \{1, \ldots, n\}$.

The cylindrical extension of the relation R to wider product space $X_1 \times \cdots \times X_n$ is the fuzzy set $\mathrm{ce}(R)$ with membership function

$$m_{\mathrm{ce}(R)}(x_1,\ldots,x_n) := m_R(x_{i_1},x_{i_2},\ldots,x_{i_r})$$

$$\mathrm{ce}(R) = (X_1 \times \cdots \times X_n, m_{ce(R)}) = \int_{X_1 \times \cdots \times X_n} m_R(x_{i_1},\cdots,x_{i_r})/(x_1,\ldots,x_n).$$

Notice that the membership funtion of the cylindrical extension is defined by the representants of the original membership function in the original variables.

We shall see later in the course of applications of fuzzy systems that after having determined the resulting fuzzy relation which is ususally defined on the base set $X_1 \times \cdots \times X_n \times Y$ we have to project the resultant relation to the output space Y of the fuzzy system. Hence it is necessary to specify the fuzzy projection. Therefore let R be a fuzzy relation on the product space $X_1 \times \cdots \times X_n$, then its projection to the $\mathrm{X}_{\mathrm{i}_1} \times \ldots \times \mathrm{X}_{\mathrm{i}_r}$ space is a fuzzy relation $P = \mathrm{Proj}_{X_{i_1} \times \cdots \times X_{i_r}}(R)$, whose membership function, with the notation $\{j_1,\ldots,j_{n-r}\} = \{1,\ldots,n\} \setminus \{i_1,\ldots,i_r\}$, is

$$m_P(x_{i_1},\ldots,x_{i_r}\} := \sup_{x_{j1},\ldots,x_{j_{n-r}}} m_R(x_1,\ldots,x_n)$$

$$\mathrm{Proj}_{X_{i_1} \times \cdots \times X_{i_r}}(R) = (X_{i_1} \times \cdots \times X_{i_r}, m_P).$$

In the case of finite base set or compact (bounded and closed) base set and continuous m_R membership function, $\sup = \max$. Notice that the membership function of the projection is defined by the most possible value of the original membership function in the cancelled variables.

1.3.4 Fuzzy join

The fuzzy join is used for fitting contiguous or disjoint relations, and also plays a significant role in fuzzy logic operations. Let *R* and *S* denote fuzzy relations:

R is a fuzzy relation on domain $X_1 \times X_2 \times \cdots \times X_r$,
S is a fuzzy relation on domain $X_m \times X_{m+1} \times \cdots \times X_n$, $m \le r+1$
$\mathrm{Join}(R,S)$ is a fuzzy relation on domain $X_1 \times X_2 \times \cdots \times X_n$.

Let us notice that in case of $m < r+1$ the relations are intersecting, while in case of $m = r+1$ the base sets are disjoint. We can obtain fuzzy join by cylindrical

extension of both relations to the product space $X_1 \times \cdots \times X_n$ then considering the intersection of the extended relations:

$$\mathrm{Join}(R,S) = \mathrm{ce}(R) \cap \mathrm{ce}(S)$$

$$m_{\mathrm{Join}(R,S)}(x_1,\ldots,x_n) := m_R(x_1,\ldots,x_r) \wedge m_S(x_m,\ldots,x_n)$$

$$\mathrm{Join}(R,S) = \int_{X_1\times\cdots\times X_n} \{m_R(x_1,\cdots,x_r) \wedge m_S(x_m,\cdots,x_n)\}/(x_1,\cdots,x_n).$$

Any *T*-norms are allowed, for example $\wedge = \min$ etc.

1.3.5 Fuzzy composition

At the algorithmization of fuzzy implications, the fuzzy composition plays an important role. Let *R* and *S* be fuzzy relations, which are defined on not the same, but adjoined product spaces. We search for the fuzzy composition $R \circ S$ of *R* and *S*, that is a relation which has its adjoined part eliminated from the full product space:

R fuzzy relation on product space $X_1 \times \cdots \times X_{m-1} \times X_m \times \cdots \times X_r$,
S fuzzy relation on product space $X_m \times \cdots \times X_r \times X_{r+1} \times \cdots \times X_n$,
$R \circ S$ fuzzy relation on product space $X_1 \times \cdots \times X_{m-1} \times X_{r+1} \times \cdots \times X_n$.

We can obtain fuzzy composition by cylindrically extending both relations to the $X_1 \times \cdots \times X_n$ product space, then considering the intersection of the extended relations and project it to the product space $X_1 \times \cdots \times X_{m-1} \times X_{r+1} \times \cdots \times X_n$.

$$R \circ S = \mathrm{Proj}_{X_1\times\cdots\times X_{m-1}\times X_{r+1}\times\cdots\times X_n}\left[(\mathrm{ce}(R) \cap \mathrm{ce}(S)\right] = \mathrm{Proj}[\mathrm{Join}(R,S)]$$

$$m_{R\circ S}(x_1,\ldots,x_{m-1},x_{r+1},\ldots,x_n) := \sup_{x_m,\ldots,x_r} m_R(x_1,\ldots,x_r) \wedge m_S(x_m,\ldots,x_n)$$

Any *T*-norms are allowed, for example $\wedge = \min$ etc.

The composition can be interpreted as the matching of two fuzzy sets. First, we extend the two fuzzy sets (ce), then match them on their common product space (sup). The result is a fuzzy set on the disjoint parts of the product space. The process is similar to a rule based system, where we match data to the condition part of a rule. The data corresponds to the common product space.

We demonstrate the computing rules on a simple example. Suppose that X, Y, Z are discretized so that $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3, y_4\}$, $Z = \{z_1, z_2, z_3\}$. Let *R* be a fuzzy relation on the $X \times Y$ set, *S* a fuzzy relation on $Y \times Z$, were

R: x is significantly greater than y,
S: y is very close to z ,

and let us assume that the membership function of each relation is specified by a table, e.g. the value of $m_R(x_1, y_1)$ is given at the intersection of x_1 and y_1. Assuming R and S are known and $\wedge = \min$, we illustrate the computation of $R \circ S$ in *Fig. 1.8.*

R	y_1	y_2	y_3	y_4
x_1	0.8	1	0.1	0.7
x_2	0	0.8	0	0
x_3	0.9	1	0.7	0.8

S	z_1	z_2	z_3
y_1	0.4	0.9	0.3
y_2	0	0.4	0
y_3	0.9	0.5	0.8
y_4	0.6	0.7	0.5

$R \circ S$	z_1	z_2	z_3
x_1	0.6	0.8	0.5
x_2	0	0.4	0
x_3	0.7	0.9	0.7

$$m_{R\circ S}(x_i, z_k) = \max_{y_j} \min\left\{m_R(x_i, y_j), m_S(y_j, z_k)\right\}$$

$$R \circ S = \sum_{X\times Z} \max_{y_j} \min\left\{m_R(x_i, y_j), m_S(y_j, z_k)\right\}/(x_i, z_k)$$

Fig. 1.8 Computation of fuzzy composition for relations over discrete sets

1.3.6 Fuzzy extension of deterministic function

The deterministic function $y = f(x_1, \ldots, x_n)$ can be interpreted as a special relation f on the domain $X_1 \times \cdots \times X_n \times Y$, which assigns a unique value y to each $(x_1, \ldots, x_n)$ (if the function is single valued). The usual notation $f : (x_1, \ldots, x_n) \to y$ is only another form of the relation $(x_1, \ldots, x_n)\, f\, y$. This approach has the capability of the fuzzy extension of the function f. Namely, let A_i be a fuzzy set on X_i, whose membership function is $m_{A_i}(x_i)$, $i = 1, \ldots, n$. Consider the fuzzy relation R, which is the direct product of the above fuzzy sets:

$$R = A_1 \times \cdots \times A_n,$$
$$m_R(x_1, \ldots, x_n) = m_{A_1}(x_1) \wedge \ldots \wedge m_{A_n}(x_n).$$

Consider now the crisp relation f on the domain $X_1 \times \cdots \times X_n \times Y$, then it can be interpreted as a fuzzy relation S, whose membership function m_S is crisp:

$$m_S(x_1,\ldots,x_n,y) = \begin{cases} 1, & \text{if } y = f(x_1,\ldots,x_n) \\ 0, & \text{otherwise.} \end{cases}$$

Hence, the fuzzy extension of the function f can be referred by the fuzzy set $f = \mathrm{ce}(R) \cap S$ (denoted also by f). According to the definition of cylindrical extension and intersection we obtain:

$$\begin{aligned} m_f(x_1,\ldots,x_n,y) &= m_{\mathrm{ce}(R)}(x_1,\ldots,x_n,y) \wedge m_S(x_1,\ldots,x_n,y) = \\ &= m_R(x_1,\ldots,x_n) \wedge m_S(x_1,\ldots,x_n,y) = \\ &= \begin{cases} m_{A_1}(x_1) \wedge \cdots \wedge m_{A_n}(x_n), & \text{if } y = f(x_1,\ldots,x_n) \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

In the classical approach we were interested in the value $y = f(x_1,\ldots,x_n) \in Y$, that corresponds now to the fuzzy set F on domain Y, which is the projection of the fuzzy function f to Y:

$$F = \mathrm{Proj}_Y(f) = \mathrm{Proj}_Y(ce(R) \cap S) = R \circ S,$$

$$\begin{aligned} m_F(y) &= \sup_{x_1,\ldots,x_n} m_f(x_1,\ldots,x_n,y) = \\ &= \begin{cases} \sup\limits_{x_1,\ldots,x_n \in f^{-1}(y)} m_{A_1}(x_1) \wedge \cdots \wedge m_{A_n}(x_n), & \text{if } f^{-1}(y) \neq \varnothing \\ 0, \text{ if } f^{-1}(y) = \varnothing \end{cases} \end{aligned}$$

Let us notice that the membership function of the fuzzy set F is chosen in such a way that $m_F(y)$ results in the most possible value satisfying the relation $y = f(x_1,\ldots,x_n)$.

1.3.7 Fuzzy dynamic systems

Time invariant nonlinear dynamic systems in discrete time can be written in the form

$$\begin{aligned} x_{n+1} &= f(x_n, u_n) \\ y_n &= g(x_n, u_n). \end{aligned}$$

We shall assume that f and g have been already extended to fuzzy functions and $D := X(n) \times U(n)$ is a fuzzy set. The next state and actual output of the fuzzy dynamic system will be fuzzy sets defined by compositions: $X(n+1) = D \circ f$ and $Y(n) = D \circ g$. Let us use the following notation:

X, U, Y	base sets for state, input and output
$X(n+1), X(n), U(n), Y(n)$	fuzzy sets
f, g	fuzzy extensions of deterministic functions
$X(n+1) = F := D \circ f$	fuzzy set of next state (projection to X)
$Y(n) = G := D \circ g$	fuzzy set of actual output (projection to Y)

Then the membership functions of $X(n+1)$ and $Y(n)$ are as follows:

$$m_F(x_{n+1}) = \begin{cases} \sup\limits_{(x_n,u_n) \in f^{-1}(x_{n+1})} m_{X(n)}(x_n) \wedge m_{U(n)}(u_n) \wedge m_f(x_n, u_n, x_{n+1}), f^{-1}(x_{n+1}) \neq \varnothing \\ 0, \text{ if } f^{-1}(x_{n+1}) = \varnothing \end{cases}$$

$$m_G(y_n) = \begin{cases} \sup\limits_{(x_n,u_n) \in g^{-1}(y_n)} m_{X(n)}(x_n) \wedge m_{U(n)}(u_n) \wedge m_g(x_n, u_n, y_n) \quad \text{if } g^{-1}(y_n) \neq \varnothing \\ 0, \text{ if } g^{-1}(y_n) = \varnothing \end{cases}$$

If u_n is crisp then $m_{U(n)}(u_n)$ can be left in the above formulas, moreover the sup is considered in x_n only. A typical simulation objective for a fuzzy dynamic system would be to determine the membership functions of $X(n+1)$ and $Y(n)$ once the membership functions of $X(n)$, f and g, and the crisp input u_n are known.

1.4 Fuzzy logic

By using the conjunction, disjunction and negation operations (commonly used in binary logic) we can logically relate linguistic variables, to which we can assign fuzzy sets to characterize their uncertainty. Since the possibility decreases (the uncertainty increases) at simultaneous occurrence and the possibility increases (the uncertainty decreases) at alternative occurrence, therefore it is obvious that we assign the fuzzy intersection $(\cap)$ to the conjunction (and) operation, the fuzzy union $(\cup)$ to the disjunction (or) operation and the fuzzy complement to the negation.

1.4.1 Fuzzy conjunction, disjunction and negation

Let x be the linguistic variable, let X be the base set, and let A, B be fuzzy sets assigned to the linguistic variable on X with membership functions m_A and m_B respectively, then the results of fuzzy conjunction, disjunction and negation are fuzzy sets:

Conjunction:

$$x \text{ is } A \text{ and } x \text{ is } B$$
$$m_{A\cap B}(x) = m_A(x) \wedge m_B(x)$$

Disjunction:

$$x \text{ is } A \text{ or } x \text{ is } B$$
$$m_{A\cup B}(x) = m_A(x) \vee m_B(x)$$

Negation:

$$x \text{ is not } A$$
$$m_{A^c}(x) = c(m_A(x))$$

The interpretation of conjunction and disjunction is a little bit more difficult if the variables have different sets as domain. Let $x_1, \ldots, x_N$ be elementary variables, and let $X_1, \ldots, X_N$ the corresponding base sets, furthermore let $\{i_1, \ldots, i_r\}$ and $\{j_1, \ldots, j_m\}$ be subsets of the index set $\{1, \ldots, N\}$, let $X_A = X_{i_1} \times \cdots \times X_{i_r}$ and $X_B = X_{j_1} \times \cdots \times X_{j_m}$ be product sets, let $x_A \in X_A$, $x_B \in X_B$ be aggregated variables and $A \subset X_A$, $B \subset X_B$ be fuzzy sets on the product spaces. Then the general forms of conjunction and disjunction are the following:

Conjunction:

$$x_A \text{ is } A \text{ and } x_B \text{ is } B = v \text{ is } P$$
$$P = \text{ce}(A) \cap \text{ce}(B)$$
$$m_P(v) = m_A(x_A) \wedge m_B(x_B)$$

Disjunction:

$$x_A \text{ is } A \text{ or } x_B \text{ is } B = v \text{ is } P$$
$$P = \text{ce}(A) \cup \text{ce}(B)$$
$$m_P(v) = m_A(x_A) \vee m_B(x_B)$$

1.4.2 Fuzzy implication

There are several methods for implications widely used in practice. As is well known, the implication operation $a \to b$ of classical binary logic can be written in different forms according to the Boolean operation (truth) table

a	b	$a \to b$
0	0	1
0	1	1
1	0	0
1	1	1

and denoting the logic operations by ·,+,¯, it can be written:

$$a \to b = a \cdot b + \overline{a} \tag{1}$$
$$a \to b = \overline{\overline{a \cdot \overline{b}}} = \overline{a} + b \tag{2}$$

Equation (1) describes the Zadeh-implication, equation (2) is the base of the Kleene-Dienes implication, while the following equation which is not compatible with Boolean algebra

$$a \to b \approx a \cdot b \tag{3}$$

is the base of the Mamdani implication, which is proved to be satisfactory in practice and easily realizable. Other implication concepts are borrowed from probability theory and multi-valued logic [5].

Kleene-Dienes implication:

$$a \to b = \overline{a} + b\,,\ S = \max$$
$$R_{KD} = ce\left(A^c\right) \cup ce(B)$$
$$m_{R_{KD}}(a,b) = \max\{1 - m_A(a), m_B(b)\}$$

Lukasiewicz-implication:

$$a \to b = \bar{a} + b\,,\ S = S_0$$
$$R_L = ce\left(A^c\right) \cup ce(B)$$
$$m_{R_L}(a,b) = \min\{1, 1 - m_A(a) + m_B(b)\}$$

Zadeh-implication:

$$a \to b = ab + \bar{a}\,,\ T = \min\,,\ S = \max$$
$$R_Z = (ce(A) \cap ce(B)) \cup ce\left(A^c\right)$$
$$m_{R_Z}(a,b) = \max\{\min\{m_A(a), m_B(b)\}, 1 - m_A(a)\}$$

Stochastic implication:

$$P(B|A) = 1 - P(A) + P(A)P(B),\quad T = H_1\,,\ S = S_0$$
$$R_* = ce\left(A^c\right) \cup \{ce(A) \cap ce(B)\}$$
$$m_{R_*}(a,b) = \min\{1, 1 - m_A(a) + m_A(a)m_B(b)\}$$

Gödel-implication:

$$\text{multi-valued logic: } v\left(a \underset{g}{\to} b\right) = \begin{cases} 1, \text{ if } & v(a) \le v(b) \\ v(b), & \text{otherwise} \end{cases}$$
$$m_{R_g}(a,b) = \left(m_A(a) \underset{g}{\to} m_B(b)\right) = \begin{cases} 1, \text{ if } & m_A(a) \le m_B(b) \\ m_B(b), & \text{otherwise} \end{cases}$$

Sharp-implication:

$$v\left(a \underset{s}{\to} b\right) = \begin{cases} 1, \text{ if } & v(a) \le v(b) \\ 0, & \text{otherwise} \end{cases}$$
$$m_{R_s}(a,b) = \left(m_A(a) \xrightarrow[s]{} m_B(b)\right)$$

General-implication:

$$a \to b = \left(a \xrightarrow[a]{} b\right) \wedge \left(\neg a \xrightarrow[b]{} \neg b\right),\ T = \min$$
$$\mathbf{a}, \mathbf{b} \in \{s, g\}$$
$$m_{R_{ab}}(a,b) = \min\left\{\left(m_A(a) \xrightarrow[a]{} m_B(b)\right), \left((1 - m_A(a)) \xrightarrow[b]{} (1 - m_B(b))\right)\right\}$$

Mamdani-implication:

$$a \to b := a \wedge b\,,\ T = \min$$
$$R_M = ce(A) \cap ce(B)$$
$$m_{R_M}(a,b) = \min\{m_A(a), m_B(b)\}$$

The fuzzy *inclusion* is defined according to the values of the membership functions, that is

$$A \supset B \Leftrightarrow \forall x : m_A(x) \geq m_B(x).$$

The following equation is valid among the various implications [5]:

$$R_L \supset R_* \supset R_{KD} \supset R_Z \supset R_M$$
$$R_L \supset R_G \supset R_M$$
$$R_G \supset R_S .$$

1.4.3 Fuzzy knowledge base

In the following, we will use Mamdani implication in most parts of this book. Let $x \in X$, $A \subset X$ be fuzzy set, and similarly let $y \in Y$ and $B \subset Y$ be fuzzy set. The simplest case of fuzzy implication is the following:

Implication:

$$if \ \ x \ \ is \ \ A \ \ then \ \ y \ \ is \ \ B$$

Mamdani implication:

$$A \to B = \mathrm{ce}(A) \cap \mathrm{ce}(B)$$
$$m_{A \to B}(x, y) = m_A(x) \wedge m_B(y)$$
$$A \to B = \int_{X \times Y} m_A(x) \wedge m_B(y) / (x, y)$$

Therefore, $R = A \to B \subset X \times Y$ is a relation on domain $X \times Y$. We can consider the effect of the relation to y that is the projection of the relation $R = A \to B$ to Y, from which we get the composition rule of fuzzy implication:

$$\mathrm{Proj}_Y(R) = \mathrm{Proj}_Y(A \to B) = A \circ B$$

where $A \circ B$ is the fuzzy composition defined earlier. Now consider the possibilities of generalization. Assume that the relation R contains an *else* branch too, then it implies from the above:

$$R : if \ \ x \ \ is \ \ A \ \ then \ \ y \ \ is \ \ B \ \ else \ \ y \ \ is \ \ C$$
$$R = (A \to B) \cup (A^c \to C)$$
$$m_R(x, y) = [m_A(x) \wedge m_B(y)] \vee [c(m_A(x)) \wedge m_C(y)]$$
$$\mathrm{Proj}_Y(R) = \mathrm{Proj}_Y([\mathrm{ce}(A) \cap \mathrm{ce}(B)] \cup [\mathrm{ce}(A^c) \cap \mathrm{ce}(\mathrm{C})]) .$$

Especially if $T = \min$, $S = \max$ and $c(a) = 1 - a$ then

$$m_R(x, y) = \max\{\min\{m_A(x), m_B(y)\}, \min\{1 - m_A(x), m_C(y)\}\}.$$

Thus, in case of practical applications it can be supposed that only *if …then…* variants are used without *else* branches. Therefore a sufficiently general fuzzy rule base can be defined in the following form, where $R_1, \ldots, R_n$ are the relations of the fuzzy rule (knowledge) base $R = \bigcup_{i=1}^{n} R_i$:

$$R_i : \textit{if } x_1 \textit{ is } X_1^i \textit{ and } \ldots \textit{ and } x_N \textit{ is } X_N^i$$
$$\textit{then } y \textit{ is } Y^i$$

The first part of the relation is the antecedent, the second part is the consequent.

In a fuzzy system, the rule base R_i, $i = 1, \ldots, n$ forms the linguistic statements of the expert knowledge, in which $x_1, \ldots, x_N, y$ are fuzzy quantities. These rules contain fuzzy *if …then* implications. The complete set of rules is the knowledge base of the fuzzy system.

1.4.4 Data matching for a single relation

Let the fuzzy set R denote the complete set of fuzzy relations, which is composed of the rules. During the application of the fuzzy system, at first the rules must be matched to the available data, which usually come from a process (context). Next, the rule matched in this manner "fires", due to develop the actuating signal (answer) for the process. Generally the context consists of crisp or fuzzy data (D) measured on the process. The data set can be considered as a fuzzy set D, because the crisp case is its degenerated case when $m_D = k_D$. Thus, the composition rule of implication:

$$C = D \circ R = D \circ \left(\bigcup_{i=1}^{n} R_i \right),$$

where C is the action in fuzzy form that we have to convert later to crisp value (defuzzification). We shall formulate the rules of fuzzy inference (approximate reasoning) as an algorithm, but first we have to clarify some details.

If A_j and B_j are fuzzy sets on the same domain X_j, X is a product set having X_j among its components, and the cylindrical extension is related to X, then

$$\mathrm{ce}(A_j) \cap \mathrm{ce}(B_j) = \mathrm{ce}(A_j \cap B_j)$$

because the integral forms of both sides are identical with each other:

$$\int_X m_{A_j}(x_j)/x \cap \int_X m_{B_j}(x_j)/x = \int_X [m_{A_j}(x_j) \wedge m_{B_j}(x_j)]/x\,.$$

In the next table the resultant membership function is denoted by bold curve in the function of variable x_j, where the position of maximum is marked by an arrow. On the left side of the figure $A_j = \{x_j^*\}$ is a crisp set, while on the right side A_j is a fuzzy set. The latter case corresponds to the approximation of a noisy x_j, where the position of the maximum of $m_{A_j}(x_j)$ is the measured value x_j^*, and the triangular shaped membership function is concentrated to the interval $[x_j^* - 3s\,, x_j^* - 3s\,]$ around the measured value, where s is the standard deviation of noise (the mean value of the additive noise was supposed to be zero). This approach can be considered to be the fuzzy triangular interpretation of the bell-shaped (Gaussian) curve. *Fig.1.9* illustrates the computation of the effect of an elementary term

$$\sup_{x_j} m_{A_j}(x_j) \wedge m_{B_j}(x_j) = \sup_{x_j} \min\{m_{A_j}(x_j), m_{B_j}(x_j)\}$$

for crisp (left) and fuzzy (right) A_j and $T = \min$, which will play an important role in fuzzy approximate reasoning.

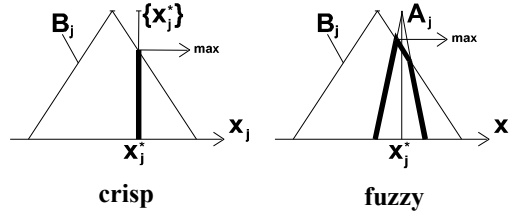


Fig. 1.9 Principle of single variable data matching (*T*=min)

Now consider the method of matching $C = D \circ R_i$ in case of a single relation R_i. Here, using the notation of the previous case, A_j corresponds to the fuzzified crisp or noisy value of the measured x_j^*, while the fuzzy set X_j^i replaces the B_j set (for

example NB,NM,…,PM,PB). The nomination and indices indicate that we deal with a fuzzy set X_j^i with domain X_j, which is included in the *i*-th relation. According to the rules of fuzzy logic the relation and data can be cylindrically extended to the common product set $X_1 \times \cdots \times X_N \times Y$. Assuming Mamdani implication the (cylindrically extended) fuzzy forms of relation R_i and input data D are as follows:

$$R_i = \mathrm{ce}(X_1^i) \cap \cdots \cap \mathrm{ce}(X_N^i) \cap \mathrm{ce}(Y^i)$$
$$m_{R_i}(x_1, \ldots, x_N, y) = m_{X_1^i}(x_1) \wedge \cdots \wedge m_{X_N^i}(x_N) \wedge m_{Y^i}(y)$$

$$D = \mathrm{ce}(A_1) \cap \ldots \cap \mathrm{ce}(A_N)$$
$$m_D(x_1, \ldots, x_N, y) = m_{A_1}(x_1) \wedge \cdots \wedge m_{A_N}(x_N)$$

Let us apply the composition rule for the single relation R_i and let us take into consideration that the *T*-norm is associative:

$$D \circ R_i = \mathrm{Pr}oj_Y(D \cap R_i)$$
$$m_{D \circ R_i}(y) = \sup_{x_1,\ldots,x_N} m_D(x_1, \ldots, x_N, y) \wedge m_{R_i}(x_1, \ldots, x_N, y) =$$
$$= \sup_{x_1,\ldots,x_N} [m_{A_1}(x_1) \wedge \cdots \wedge m_{A_N}(x_N)] \wedge [m_{X_1^i}(x_1) \wedge \cdots \wedge m_{X_N^i}(x_N) \wedge m_{Y^i}(y)] =$$
$$= \sup_{x_1,\ldots,x_N} [m_{A_1}(x_1) \wedge m_{X_1^i}(x_1)] \wedge \cdots \wedge [m_{A_N}(x_N) \wedge m_{X_N^i}(x_N)] \wedge m_{Y^i}(y) =$$
$$= [\sup_{x_1} m_{A_1}(x_1) \wedge m_{X_1^i}(x_1)] \wedge \cdots \wedge [\sup_{x_N} m_{A_N}(x_N) \wedge m_{X_N^i}(x_N)] \wedge m_{Y^i}(y)$$

We can introduce the fireing weights

$$t_{ij} := \sup_{x_j} m_{A_j}(x_j) \wedge m_{X_j^i}(x_j)$$
$$t_i := t_{i1} \wedge \cdots \wedge t_{iN},$$

then

$$m_{D \circ R_i}(y) = t_i \wedge m_{Y^i}(y).$$

Let us consider the following two cases which are often used in the practice:

T=min: $t_{ij} := \sup_{x_j} \min\{m_{A_j}(x_j), m_{X_j^i}(x_j)\}$, $t_i := \min\{t_{i1}, \cdots, t_{iN}\}$,

T=dot: $t_{ij} := \sup_{x_j} m_{A_j}(x_j) \cdot m_{X_j^i}(x_j)$, $t_i := t_{i1} \cdot \ldots \cdot t_{iN}$.

Fig. 1.10 illustrates the case *T*=min.

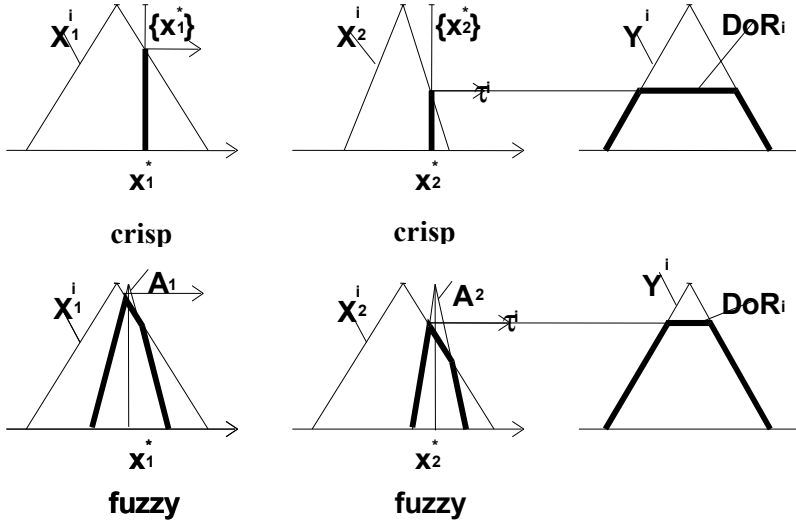


Fig. 1.10 Multiple variable data matching in the case of a single relation (*T*=min)

1.4.5 Composition based inference

Consider now the general (and practical) case, where there are multiple rules $(R_1,\ldots,R_n)$ given in the knowledge base, and the union (or) of them means the resulting rule from which the output of the fuzzy system has to be resolved:

Fuzzy knowledge base: $R=\bigcup_{i=1}^{n} R_i$

Relation R_i : $\underbrace{\textit{if } x_1 \textit{ is } X_1^i \textit{ and } \ldots \textit{ and } x_N \textit{ is } X_N^i}_{\text{antecedent}}$ $\textit{then } \underbrace{y \textit{ is } Y^i}_{\text{consequent}}$

Antecedent:

$$B^i := \mathrm{ce}(X_1^i) \cap \cdots \cap \mathrm{ce}(X_N^i)$$

$$m_{B^i}(x_1,\ldots,x_N,y) = m_{X_1^i}(x_1) \wedge \cdots \wedge m_{X_N^i}(x_N)$$

Implication:

$$R_i = B^i \rightarrow Y^i$$

$$m_{R_i}(x_1,\ldots,x_N,y) = m_{B^i \rightarrow Y^i}(x_1,\ldots,x_N,y)$$

Observation (data):

$$x_j^* \rightarrow \text{crisp } A_j \text{ or fuzzy } A_j$$

$$D = \mathrm{ce}(A_1) \cap \ldots \cap \mathrm{ce}(A_N)$$

$$m_D(x_1,\ldots,x_N,y) = m_{A_1}(x_1) \wedge \cdots \wedge m_{A_N}(x_N)$$

Composition based inference:

$$D \circ R = D \circ \bigcup_{i=1}^{n} R_i = \mathrm{Proj}_Y \left\{ D \circ \bigcup_{i=1}^{n} R_i \right\}$$

$$m_{D\circ R}(y) = \sup_{x_1,\ldots,x_N} m_D(x_1,\ldots,x_N,y) \wedge \left\{ \bigvee_{i=1}^{n} m_{B^i \to Y^i}(x_1,\ldots,x_N,y) \right\}$$

Unfortunately the evaluation of $m_{D\circ R}(y)$ for general implication is a hard problem because the order of $\sup, \wedge, \vee, B^i \to Y^i$ can not be changed. However for Mamdani implication, *S*=max and *T*=min or *T*=dot simple algorithms can be derived. For example for *T*=min yields:

$$\begin{aligned} m_{D\circ R}(y) &= \sup_{x_1,\ldots,x_N} m_D(x_1,\ldots,x_N,y) \wedge \left\{ \bigvee_{i=1}^{n} [m_{B^i}(x_1,\ldots,x_N,y) \wedge m_{Y^i}(y)] \right\} = \\ &= \sup_{x_1,\ldots,x_N} \min\{\min\{m_{A_1}(x_1),\ldots\}, \max_{1\le i\le n}\{\min\{\min\{m_{X_1^i}(x_1),\ldots\}, m_{Y^i}(y)\}\}\} = \\ &= \max_{1\le i\le n}\{\min\{\min\{\sup_{x_1} \min\{m_{A_1}(x_1), m_{X_1^i}(x_1)\},\ldots\}, m_{Y^i}(y)\}\} = \\ &= \max_{1\le i\le n} m_{D\circ R_i}(y) \end{aligned}$$

or in equivalent form

$$m_{D\circ R_i}(y) = \min\{\min_j t_{ij}, m_{Y^i}(y)\} = \min\{t_i, m_{Y^i}(y)\}$$

$$m_{D\circ R}(y) = \max_i m_{D\circ R_i}(y).$$

Similar result can be derived for Mamdani implication, *S*=max and *T*=dot norms:

$$m_{D\circ R_i}(y) = (\prod_j t_{ij}) \cdot m_{Y^i}(y) = t_i \cdot m_{Y^i}(y)$$

$$m_{D\circ R}(y) = \max_i m_{D\circ R_i}(y).$$

In both cases the earlier notations for t_{ij} and t_i were used. Notice that $\max_i m_{D\circ R_i}(y)$ is the upper cover of the $m_{D\circ R_i}(y)$ functions and therfore it can easily be determined.

Composition rule of Mamdani-type fuzzy inference (S=max, T=min or T=dot):

$$\mu_{D\circ R}(y) = \max_i \mu_{D\circ R_i}(y)$$

Mamdani-type max-min inference algorithm:

1) Evaluate the value of t_{ij}, $j = 1,\ldots,N$ and t_i for each rule R_i, and evaluate the value of function $\mu_{D\circ R_i}(y)$ for each y:

$$t_{ij} = \begin{cases} m_{X_j^i}(x_j^*), & \text{if } x_j \text{ is crisp} \\ \sup_{x_j} \min\left\{m_{A_j}(x_j), m_{X_j^i}(x_j)\right\}, & \text{otherwise} \end{cases}$$

$$t_i = \min_j t_{ij}$$

$$m_{D\circ R_i}(y) = \begin{cases} m_{Y^i}(y), & \text{if } m_{Y^i}(y) \le t_i \\ t_i, & \text{otherwise} \end{cases}$$

2) Evaluate the membership function of the output signal of the fuzzy system with max operation for all y:

$$m_{D\circ R}(y) = \max_i m_{D\circ R_i}(y)$$

Mamdani-type max-dot inference algorithm:

1) For each rule R_i apply the *T*=dot($\wedge$) (product) *T*-norm, with unchanged *S*=max($\vee$) *S*-norm:

$$t_{ij} = \begin{cases} m_{X_j^i}(x_j^*), & \text{if } x_j \text{ is crisp} \\ \sup_{x_j} \left\{m_{A_j}(x_j) \cdot m_{X_j^i}(x_j)\right\}, & \text{otherwise} \end{cases}$$

$$t_i = t_{i1} \wedge t_{i2} \wedge \cdots \wedge t_{iN} = t_{i1} \cdot t_{i2} \cdot \ldots \cdot t_{iN}$$

$$m_{D\circ R_i}(y) = t_i \wedge m_{Y^i}(y) = t_i \cdot m_{Y^i}(y)$$

2) Evaluate the membership function of the output signal of the fuzzy system with max operation for all y:

$$m_{D\circ R}(y) = \max_i m_{D\circ R_i}(y)$$

It can be seen well, that the only difference between the max-dot algorithm and the above max-min algorithm is that $\wedge$=min was replaced by $\wedge$=dot. For instance in case of crisp input, t_{ij} is evaluated in the same way, t_i is the product of the individual t_{ij}'s, and $m_{D\circ R_i}$ is not computed by truncation at t_i, but originated from m_{Y^i}, simply by multiplying by t_i (less than one) at every y (proportionally scaled down).

Further simplification is possible if we replace the norm S=max to the $S_o(a,b)=\min\{1, a+b\}$ bounded sum norm. In practice further simplification might be applied by using the $a \vee b \approx a+b$ approximation. Of course this would be in contravention of the rules, since in this case the membership function could be increased to a value greater than one, however this problem can be solved during defuzzification.

Simplified sum-dot inference algorithm:

1) For each rule R_i apply the T=dot($\wedge$) and S≈sum($\vee$) norms, and calculate the value of t_{ij}, t_i and $m_{D\circ R_i}(y)$ similarly to the max-dot implication, e.g. $m_{D\circ R_i}(y) = t_i \cdot m_{Y^i}(y)$.

2) $$m_{D\circ R}(y) = \bigvee_i m_{D\circ R_i}(y) \approx \sum_i m_{D\circ R_i}(y) = \sum_i t_i \cdot m_{Y^i}(y).$$

We mention here that if the defuzzified value of the output is computed by the center of gravity (COG) method which is also known as the center of area (COA) method, then:

$$y^* = \frac{\int y m_{D\circ R}(y)dy}{\int m_{D\circ R}(y)dy} = \frac{\int y \sum_i t_i m_{Y_i}(y)dy}{\int \sum_i t_i m_{Y_i}(y)dy} = \frac{\sum_i t_i y_i^* \int m_{Y_i}(y)dy}{\sum_i t_i \int m_{Y_i}(y)dy} =$$

$$= \frac{\sum_i (t_i \int m_{Y_i}(y)dy) y_i^*}{\sum_i (t_i \int m_{Y_i}(y)dy)} = \frac{\sum_i t_i^* y_i^*}{\sum_i t_i^*}$$

where $\int m_{Y^i}(y)dy$ and $y_i^* = \int y m_{Y^i}(y)\,dy \Big/ \int m_{Y^i}(y)\,dy$ may be pre-computed. Of course $t_i^* = t_i \int m_{Y^i}(y)dy$ is input data dependent because the fireing weight t_i depends on the input data data.

Other composition based inference algorithms

Referring to the wide possibilities of the norms $S(\vee)$ and $T(\wedge)$ and implications (Kliene-Dienes, Lukasiewicz, Mamdani, etc.), we must emphasize that the derivation of the above well rendible rule $m_{D\circ R} = \underset{i}{\vee} m_{D\circ R_i}$ assumed that the T-norm is of min-type or dot-type, the S-norm is of max-type or sum-type, and the implication is of Mamdani-type, thus the validity of the above well rendible formula was proven only for these norm-combinations. However, we must mention that these cases are dominant in practical applications. Unfortunately for the general case such simple forms, which allow the superposition of the effects of the different rules, are not available.

1.4.6 Individual-rule based inference

In the general case the evaluation of the knowledge base (relations) by using composition based inference may be a hard problem which does not result in an easily computable and closed formula. Hence people often choose as inference rule the individual-rule based inference which is defined by

$$m_{D\circ R}(y) := \underset{i=1}{\overset{n}{\vee}} m_{D\circ R_i}(y)$$

The individual-rule based inference allows the superposition of the effect of the single rules which simplifies the computation.

For Mamdani-type max-min, max-dot and max-sum inference the composition based and the individual-rule based inference methods are obviously equivalent, but for general norms and implication rules these two inference methods usually express different inference strategies.

1.5 Defuzzification methods

Let us deal now with the question of defuzzification. Maintaining the notations used in the section on fuzzy logic (and not confined to the case of control only, where $y = u$), we know that after the application of the inference rules we obtain the output in the fuzzy form $m_{Y_\circ}(y) = m_{D\circ R}(y)$, which is a function of y. A typical form in case of two rules is shown in *Fig. 1.11*.

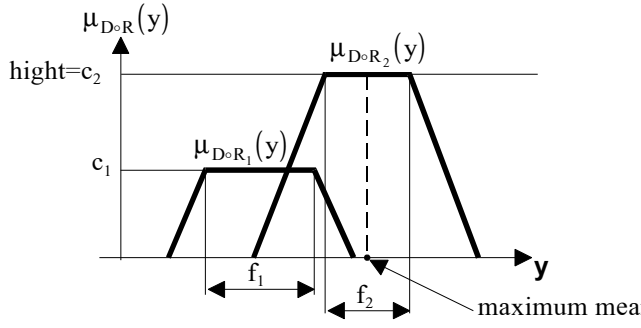


Fig. 1.11 Typical membership function of fuzzy output

The task of defuzzification is to determine the most possible crisp numerical value y^* in some sense based on the membership function $m_{D\circ R}(y)$. Some typical methods are the following:

Center of gravity (center of area) method

The method is called COG (center of gravity) or COA (center of area) method in the literature, since if there is a mass distribution $dm = r(y)\,dy$ in direction y, then $m = \int r(y)\,dy$ is the resulting mass, and the center of gravity y_c can be computed by the equation $my_c = \int yr(y)\,dy$. On the analogy of that:

$$y^*_{COA} = \frac{\int y m_{D\circ R}(y) dy}{\int m_{D\circ R}(y) dy},$$

or in discrete case

$$y^*_{COA} = \frac{\sum_j y_j m_{D\circ R}(y_j)}{\sum_j m_{D\circ R}(y_j)}.$$

Let us notice that the method does not take into consideration that the common parts of $m_{D\circ R_1}(y)$ and $m_{D\circ R_2}(y)$ are overlapped. The method might be slow in real-time applications, because the defuzzification cannot be computed separately from the

components $m_{D\circ R_i}(y)$. The integral is the common (Riemann) integral, which is denoted by dy.

Center of sums method

It is called COS (center of sums) method in the literature. The overlapped areas are considered with multiplicity:

$$y^*_{COS} = \frac{\int y \sum_i m_{D\circ R_i}(y)dy}{\int \sum_i m_{D\circ R_i}(y)dy} = \frac{\sum_i \int y m_{D\circ R_i}(y)dy}{\sum_i \int m_{D\circ R_i}(y)dy}.$$

The computation can be performed separately for each relation and the results can be summed after this. In the discrete case $\int$ is replaced by $\sum_j$.

Bisector of area method

Denote a and b the minimum and maximum value of y, respectively, for which $m_{D\circ R}(y)$ is nonzero. Then the bisector of area (BOA) is y^*_{BOA} defined by

$$\int_a^{y^*_{BOA}} m_{D\circ R}(y)\,dy = \int_{y^*_{BOA}}^{b} m_{D\circ R}(y)\,dy\,.$$

Height method

Let the maximum of $m_{D\circ R_i}(y)$ be denoted by c_i and let denote $\widetilde{f}_i$ the place in y of the center of its plateau f_i (or alternatively $\widetilde{f}_i := \arg\max Y^i(y)$), then

$$y^*_{height} = \frac{\sum_i c_i \widetilde{f}_i}{\sum_i c_i}.$$

Middle of maxima method

It is called MOM (middle of maxima) method in literature, because it takes the middle of maxima (the middle of the plateau) of $m_{D\circ R}(y)$:

$$hight = \max_y m_{D\circ R}(y)$$

$$y^*_{MOM} = \frac{\inf\{y : m_{D\circ R}(y) = \text{hight}\} + \sup\{y : m_{D\circ R}(y) = \text{hight}\}}{2}$$

Similarly we can choose the smallest of maximum ($y^*_{SOM} = \inf\{y : m_{D\circ R}(y) = \text{hight}\}$) or the largest of maximum ($y^*_{LOM} = \sup\{y : m_{D\circ R}(y) = \text{hight}\}$.

1.6 Sugeno-type fuzzy systems

The TSK- (Takagi-Sugeno-Kong) type or most often known as Sugeno-type fuzzy system unifies the principles of fuzzy and classical systems in such a manner which allows specifying deterministic consequence at the place of the consequence part of the various relations [7],[8]. This allows to use for example different deterministic algorithms in various operating points. Of course in this case the operating point is not crisp, but fuzzy, thus various operating points can exist simultaneously with different possibilities. The knowledge base (rule base) of a TSK-type fuzzy system consists of relations which have the following form:

$$R_i : \textit{ if } x_i \textit{ is } X_1^i \textit{ and } \cdots \textit{ and } x_N \textit{ is } X_N^i \textit{ then } y = f_i(x_1, \ldots, x_N), \quad i = 1, \ldots, n.$$

If $f_i = c_i$ is a constant function then the fuzzy system is a zero order Sugeno system. If $f_i = c_{i1}x_1 + \cdots + c_{iN}x_N + c_{i0}$ is a linear function then the fuzzy sytem is called first order Sugeno system.

TSK inference and defuzzification algorithm:

1) Inference:

$x_1^*, x_2^*, \ldots, x_N^*$ are crisp inputs

$$\forall R_i \mapsto \left(t_i, y_i^*\right)$$

$$t_i = m_{X_1^i}\left(x_1^*\right) \wedge \cdots \wedge m_{X_N^i}\left(x_N^*\right)$$

$$y_i^* = f_i\left(x_1^*, \cdots, x_N^*\right)$$

2) Defuzzification:

$$y^*_{TSK} = \frac{\sum_i t_i y_i^*}{\sum_i t_i}.$$

1.7 Fuzzy logic controllers

The fuzzy logic controller (FLC) is a fuzzy system which play the role of the controller [4]. The output of the FLC is the input of the actuator. The actuator output is the process input, hence the controller can modify the process behaviour. The measured process output can be fed-back to the controller and compared with the command (reference) signal. The error (together with the measured or estimated state variables) can deal as input for the FLC.

1.7.1 The structure of the fuzzy logic controller

The block diagram of a typical fuzzy controller is shown in *Fig. 1.12*.

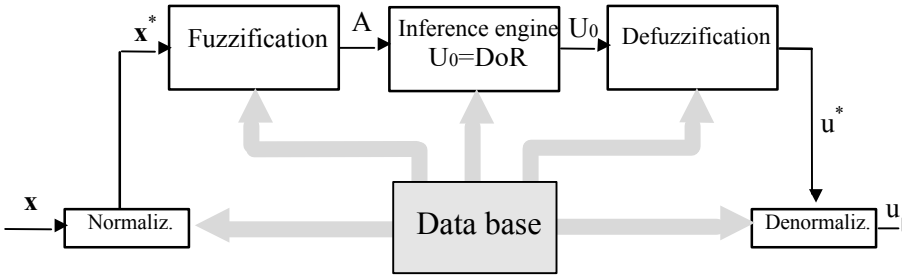


Fig. 1.12 Block diagram of a typical fuzzy logic controller (FLC)

The signals of the process (error signals, signals of sensors or observers, etc.) are represented by the vector $\mathbf{x}$. The purpose of normalization is to provide that the membership function transforms the process signal x_i to (in most case) normalized domain: $x_i^*=N_{xi}x_i$. The normalization in the system corresponds to the gain factor. The objective of fuzzification is to assign a fuzzy set A_i to the crisp or noisy normalized variable x_i^*. The aggregation of these formulates the input data D of the inference engine. The inference engine performs the inference based on the input data and the rules describing the behavior of the controller (rule base), which results in a fuzzy set U_0 (and its membership function $m_{U_0}(u)$). The purpose of defuzzification is that interpreting the fuzzy set U_0, create normalized crisp controller output u^*, which, after denormalization (gain factor) $u = N_u u^*$, gives the controller output u. Since the inference rules use several nonlinear functions (max,

min, etc.), therefore the fuzzy controller is a nonlinear controller and the effect of the normalization to the output is also nonlinear (except the effect of N_u).

We demonstrate the effect of normalization by a PID-like fuzzy controller. Let the error signal be the deviation of the reference signal and observed signal: $e = y_d - y$, then the PID algorithm is defined as

$$u = K_P\, e + K_I \int e dt + K_D\, \dot{e}\ .$$

The integral and the derivative of error can be approximated by the rectangle rule and difference quotient, respectively. Then with sampling time T we obtain

$$\int e dt = T\sum_{i=1}^{k} e(iT) = T\sum_{i=1}^{k} e_i = TSe_k$$

$$\dot{e} = \frac{e(kT) - e([k-1]T)}{T} = \frac{e_k - e_{k-1}}{T} = \frac{1}{T}\Delta e_k$$

$$u_k = K_P e_k + K_I T\Sigma e_k + \frac{K_D}{T}\Delta e_k$$

$$N_u u_k^* = \frac{K_P}{N_e} e_k^* + \frac{K_I T}{N_{\Sigma e}}\Sigma e_k^* + \frac{K_D}{TN_{\Delta e}}\Delta e_k^*$$

$$u_k^* = \frac{K_P}{N_u N_e} e_k^* + \frac{K_I T}{N_u N_{\Sigma e}}\Sigma e_k^* + \frac{K_D}{TN_u N_{\Delta e}}\Delta e_k^*\ ,$$

or neglecting the index k (the index of sample),

$$u^* = K_{PN} e^* + K_{IN}\Sigma e^* + K_{DN}\Delta e^*,$$

$$K_{PN} = \frac{K_P}{N_u N_e},\ K_{IN} = \frac{K_I T}{N_u N_{\Sigma e}},\ K_{DN} = \frac{K_D}{TN_u N_{De}}.$$

In a fuzzy-type PID controller, in which the inference rules are based on the linguistic values of the normalized variables, $K_{PN} = K_{IN} = K_{DN} = 1$ can be chosen, because any PID controller parameters

$$K_p = N_u N_e,\quad K_I = N_u N_{S\,e}/T,\quad K_D = N_u N_{De} T$$

can be reached by the appropriate choice of the normalization factors N_e, N_{Se}, N_{De}, N_u and the normalization interval $[-a^*, a^*]$.

We can choose any of the following two strategies:

i) We formulate the rules of the FLC in the natural domain of the variables x_i, u without the application of normalization and denormalization (in this case we must take into consideration the real technical limits of the variables during the elaboration of the rules).

ii) We formulate the rules of the FLC in the standardized domain of the variables x_i^*, u^* (which may be $[-1,1]$ or $[0,1]$) and apply normalization and denormalization gains. (In this case we do not need to take into consideration the technical limits of the variables during the elaboration of the rules, but we must find appropriate gain factors for the normalization and the denormalization.)

The database contains the normalization parameters, the parameters and rule of fuzzification, the parameters of membership functions, the inference rule base (the fuzzy knowledge base), the method of defuzzification and the parameters of denormalization.

The structure of fuzzy controller involves the possibility that a higher level (fuzzy or other) controller modifies the database according to the working experiences, hereby improving the quality of the controller. In such a case we talk about adaptive fuzzy control. A simpler version of this is altering the parameters (STFC=self-tuning fuzzy control), and an advanced version is altering the rules (SOFC=self-organizing fuzzy control).

In the following we show some simple examples from the area of fuzzy control.

1.7.2 PID-like fuzzy logic controller

Let y_d, y and e denote the actual value of reference signal, the process output and the error signal respectively, then (according to the custom wide spread convention in Europe) $e = y_d - y$ is the error signal (but be careful because in the USA $e = y - y_d$ is more typical which can cause problems in the interpretation of fuzzy rules in publications).

A simple (deterministic, non fuzzy) analogue PID controller has the form

$$u = K_P e + K_I \int e dt + K_D \frac{de}{dt},$$

whose discrete time (sampled data) approximation has been already discussed earlier. By using the notation $k_p = K_p$, $k_I = K_I T$ and $k_D = K_D / T$, we can write

$$u_k = k_p e_k + k_I S e_k + k_D D e_k ,$$

$$u_{k-1} = k_p e_{k-1} + k_I S e_{k-1} + k_D D e_{k-1} .$$

Consider that $Se_k - Se_{k-1} = e_k$ and $De_k - De_{k-1} = e_k - 2e_{k-1} + e_{k-2}$, then the increment $Du_k = u_k - u_{k-1}$ can be written in the following form:

$$\begin{aligned} Du_k &= k_p (e_k - e_{k-1}) + k_I e_k + k_D (e_k - 2e_{k-1} + e_{k-2}) = \\ &= (k_p + k_I + k_D) e_k + (-k_p - 2k_D) e_{k-1} + k_D e_{k-2} \end{aligned}$$

$$Du_k = k_0 e_k + k_1 e_{k-1} + k_2 e_{k-2} .$$

Thus, the implication rule of a PID-like fuzzy logic controller (PID-like FLC) has the form:

$$R_i : \textit{if } e_k \textit{ is } E_k^i \textit{ and } e_{k-1} \textit{ is } E_{k-1}^i \textit{ and } e_{k-2} \textit{ is } E_{k-2}^i \textit{ then } Du_k \textit{ is } DU_k^i .$$

Considering the fact that during the inference (max, min, etc.) and defuzzification (COA, etc.) nonlinear operations are used, we obtain a PID-like but nonlinear fuzzy logic controller

$$Du_k = F(e_k, e_{k-1}, e_{k-2}) ,$$

$$u_k = u_{k-1} + Du_k ,$$

see *Fig. 1.13*.

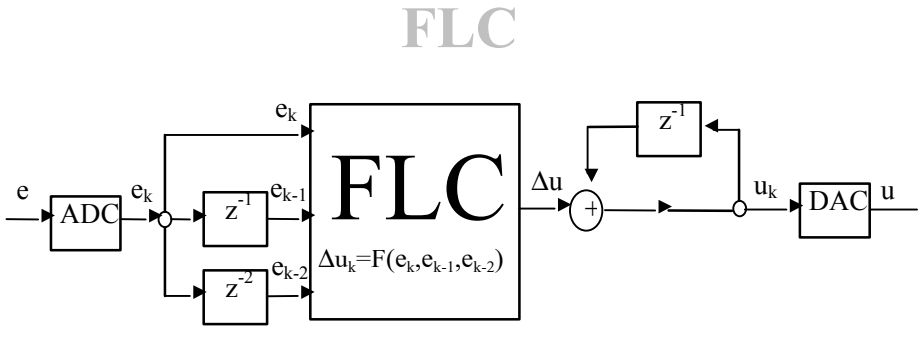


Fig. 1.13 PID-like fuzzy logic controller

Notice that the PID-like FLC (consisting of normalization, fuzzification, inference engine, defuzzification, denormalization) results in a three-argument functional relationship $F(e_k, e_{k-1}, e_{k-2})$, which requires memory too at its input and output. Like in every sampled-data controller, analogue/digital (ADC) and digital/analogue (DAC) converters are also required (the latter realizes zero order hold too). The FLC is a program or a dedicated digital hardware (e.g. a fuzzy chip), whose operation is scheduled by a real-time clock.

Since in case of three variables there are a large number of possible rules which can hardly be constructed, therefore in practice PD- and PI-like fuzzy controllers are more typical.

1.7.3 PD-like FLC

A simple sampled-data PD controller uses the algorithm

$$u_k = k_p e_k + k_D De_k .$$

If the extremums of the reference signal and the output of the plant are equal and the operating domains are symmetric, then $e_{\max} = y_{d,\max} - y_{\min}$, $e_{\min} = y_{d,\min} - y_{\max} = -e_{\max}$, $De_{\max} = e_{\max} - e_{\min}$, $De_{\min} = e_{\min} - e_{\max} = -De_{\max}$. Denoting the values of normalized variables and the boundaries of their domains by asterisk (*), it can be written that $e \in [-a_e, a_e]$, $e^* \in [-a_e^*, a_e^*]$, $?e \in [-a_{?e}, a_{?e}]$, $?e^* \in [-a_{?e}^*, a_{?e}^*]$, $u \in [-a_u, a_u]$, $u^* \in [-a_u^*, a_u^*]$. If the domains of operation are known and we want to use normalized domains, then from the relations $a_e^* = N_e a_e$, $a_{?e}^* = N_{?e} a_{?e}$, and $a_u = N_u a_u^*$ the normalization gains can be determined: $N_e = a_e^*/a_e$, $N_{?e} = a_{?e}^*/a_{?e}$ and $N_u = a_u/a_u^*$.

We notice that if we specify the normalization gains not according to the above, but by setting the normalization factor with a different method (e.g. by a STFC controller) while the normalization domains remain pre-fixed, then two problems can arise: i) the domain of operation is mapped on only to a small part of the normalized domain or ii) before reaching the boundary of the domain of operation the normalized output knocks against the boundary of the normalized domain. These are in connection with the sensitivity and the saturation of the controller. The careful choice of the normalizing gains is crucial, because in the first case the total power of the actuator can not be used, while in the second case the controller is practically a two-point (discontinuous) controller.

Suppose that we have already chosen the appropriate values of the normalization gains, and concentrate on the elaboration of the relations R_i. At the specification of the rule base, three meta-rules are advised by MacVicar-Whelan [6]:

MR1) If the error e_k and the change of error De_k are zero, then maintain the present control setting.

MR2) If the error e_k is tending to zero at a satisfactory rate, then maintain the present control setting.

MR3) If the error e_k is not self-correcting, then control action Du_k is not zero and depends on the sign and magnitude of e_k and De_k.

We develop the following train of thought. Consider the analogue PD controller (with normalized variables, but we omit the asterisk notation, since we usually do not emphasize it neither in rules). Let the normalized gain factors be $K_{PN}, K_{DN} = 1$. Let us use the notation $s = u$ and consider the "switching curve" $s = 0$ in the $(\dot{e}, e)$ state space: $s = e + \dot{e} = 0 \Rightarrow \dot{e} = -e$.

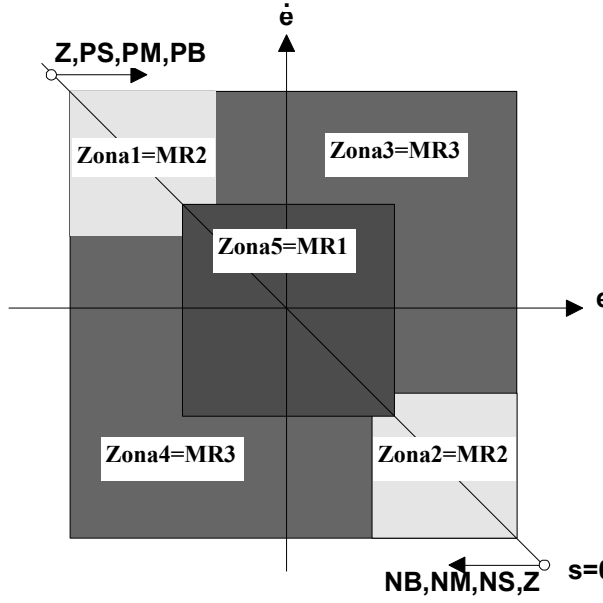


Fig. 1.14 MacVicar-Whelan zones of a PD-like FLC

In *Fig. 1.14* we labeled the switching curve $s = 0$, and the MacVicar-Whelan zones and the meta-rules related to the zones which were interpreted by the magnitude and sign of $e, \dot{e}$. According to the rule, in the area of Zone5 e and $\dot{e} \sim De$ are small, therefore MR1 becomes operative. In case of Zone1 and Zone2

the error converges to zero with a satisfactory rate, therefore MR2 is the operating rule. In case of Zone3 and Zone4, error is not self-correcting, therefore MR3 is the operating rule.

At the specification of implication rules we naturally might prefer symmetry. Because of the interpretation of negative feedback and $e = y_d - y$, we have to assure $u > 0$ above the switching curve $s = 0$, and $u < 0$ below the switching curve, respectively. Along the switching curve, e and $\dot{e} \sim De$ have equal order of magnitudes and their signs are the opposite, therefore $u = e + \dot{e} = 0$ can be chosen, which corresponds to the fuzzy value Z. Since in the top-right corner of the figure $e = \dot{e} \sim De$ is not self-correcting ($\dot{e} \sim De$ further increases the error), therefore, advancing from the left-top corner of the figure toward the right-top corner, gradually Z,PS,PM,PB controller change should be prescribed (with equal fuzzy values along the diagonals parallel with the switching curve $s = 0$). Similarly, advancing from the bottom-right corner toward the bottom-left corner, gradually Z,NS,NM,NB fuzzy values should be prescribed. For each E^i and DE^i fuzzy value pair, the fuzzy relation

$$R_i : \textit{if } e \textit{ is } E^i \textit{ and } De \textit{ is } DE^i \textit{ then } u \textit{ is } U^i$$

can be defined in a table by giving the value U^i at the intersection of column E^i and row DE^i, see *Fig. 1.15*.

	PB	Z	PS	PM	PM	PB	PB	PB
	PM	NS	Z	PS	PM	PM	PB	PB
	PS	NM	NS	Z	PS	PM	PM	PB
Δe	Z	NM	NM	NS	Z	PS	PM	PM
	NS	NB	NM	NM	NS	Z	PS	PM
	NM	NB	NB	NM	NM	NS	Z	PS
	NB	NB	NB	NB	NM	NM	NS	Z
		NB	NM	NS	Z	PS	PM	PB
		e						

Fig. 1.15 Rule base of a PD-like FLC

1.7.4 PI-like FLC

A classic sampled-data PI controller uses the algorithm

$$u_k = k_p e_k + k_I S e_k$$

whose output equations can be written in the increment form:

$$Du_k = k_p De_k + k_I e_k$$

$$Du_k = k_0 e_k + k_1 De_k$$

$$u_k = u_{k-1} + Du_k$$

If the operating domain of the change of output is $[-a_{?u}, a_{?u}]$, its normalized domain is $[-a^*_{?u}, a^*_{?u}]$, then, because of $a_{?u} = N_{?u} a^*_{?u}$, the denormalization gain will be $N_{?u} = a_{?u}/a^*_{?u}$. The PI-like FLC can be designed similarly to the PD-like one, the fuzzy rules are those in *Fig. 1.15,* but the interpretation of the defuzzified output is Du_k.

1.7.5 Supervisory fuzzy expert

At controlling complex non-linear systems we often use multi-level control algorithms. For instance, in case of robots, the non-linear dynamic model of the robot is in the form

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{t}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ are the joint variable and its derivatives respectively, $\mathbf{t}$ is the driving torque of the robot links (after gear reduction), $\mathbf{H}$ is a configuration $(\mathbf{q})$ dependent generalized inertia matrix and $\mathbf{h}$ is the state $(\mathbf{q}, \dot{\mathbf{q}})$ dependent centripetal, Coriolis, gravitational and (viscous and Coulomb) friction effects. In the model, $\mathbf{H}$ and $\mathbf{h}$ are dependent on the mass of links, on their center of gravity vector, on their (symmetric) inertia matrix, and on the corresponding parameters of the variable load. A modern control method is the non-linear decoupling in the space of joint variables $\mathbf{q}$ (called the method of computed torques control, CTC). The control algorithm consists of the centralized non-linear CTC control and the decentralized linear PID parts. The output of the non-linear part is the torque $\mathbf{t}$, and it uses the u_i outputs of the decentralized parts inside. The outputs of the decentralized PID controllers can be collected in the vector $\mathbf{u} = (u_1, u_2, \cdots)^T$. If the centralized part actuates according to the CTC control law

$$\mathbf{t} := \mathbf{H}(\mathbf{q})\mathbf{u} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}),$$

then the resultant closed loop control system $\ddot{\mathbf{q}} = \mathbf{u} \Rightarrow \ddot{q}_i = u_i$ consists of decoupled double integrators. Hence the decentralized part can be chosen in the form

$$u_i := \ddot{q}_{ai} + K_{Pi}(q_{di} - q_i) + K_{Ii}\int (q_{di} - q_i)dt + K_{Di}(\dot{q}_{di} - \dot{q}_i)$$

where $\dot{q}_{di}$ and $\ddot{q}_{di}$ are the (time-dependent) joint velocity and joint acceleration, respectively, along the desired path. In reality the centralized part only knows the $\hat{\mathbf{H}}(\mathbf{q})$ and $\hat{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}})$ nominal model belonging to the nominal load, because it has no information about the variable load:

$$\mathbf{t} := \hat{\mathbf{H}}(\mathbf{q})\mathbf{u} + \hat{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}}) .$$

In the algorithm of the decentralized part a deterministic (non-fuzzy) PID controller is included for each joint (q_i) in order to provide robustness against the change of the parameters of the load, because the system is only approximately decoupled for variable load.

Consequently it is obvious that the parameters K_{Pi}, K_{Ii}, K_{Di} of the deterministic PID controllers may be tuned by fuzzy experts in order the system be able to protect itself from the strongly increasing errors in case of variable load. In the following we will outline the method of *deSilva and MacFarlaine* [10] for tuning the parameters of the PID controllers. According to the experience, this hierarchic control (based on centralized and decentralized deterministic control and a higher level fuzzy expert system) can assure the high accuracy (0.01 ... 0.1 mm) expected with robots even in case of variable load and high speed. This prescribed accuracy cannot be assured by purely fuzzy principle-based simple controllers only.

In case of fuzzy tuning, each component $e_i = q_{di} - q_i$ of the control error is processed by a special prefilter. The interval of processing can be an integer multiple of the sampling time T of the controller (T is approx. 1 ms in case of robots). The prefilter analyses the error component in every evaluation period. The output of the prefilter is the extremum in the evaluated interval. Let A, B, C denote the extrema of the last three evaluations, where C is the latest value. The output of the prefilter is evaluated by an expert, which concludes to such fuzzy terms as accuracy (ACC), oscillation (OSC), convergence (CON), divergence (DIV) and offset (OFF). We use three special parameters for the evaluation of these attributes: the tolerance of error $(e_{\max})$, the amplitude of oscillation (a_{osc}) and the velocity of acceptable convergence $(l_{\min})$. The expert assigns fuzzy values to the error attributes and to the variables of the parameter change actions of the controller. It evaluates the fuzzy values of the controller parameters based on the rules taken from its database,

defuzzifies its values and gives them for processing to the decentralized PID controller.

Since the principle of controller tuning is identical for all degree of freedom (q_i), thus in the following we will neglect the index i. The prefilter and the fuzzy expert use discrete (quantized) variables. If the control is accurate (ACC=0), then the tuning of controller parameters is suspended, else (ACC=1) the tuning of controller parameters is processed based on the rules. The block diagram of the prefilter is shown in *Fig. 1.16.*

Fig. 1.16 Block diagram of the prefilter

The outputs of the prefilter are determined by the following principle:

1) ACC=0, if |A|, |B|, |C|< $e_{\max}$, otherwise ACC=1.

2) OSC_PRESENT=1, if
((B<A-2a_{osc}) and (C>B+2a_{osc})) or
((B>A+2a_{osc}) and (C<B-2a_{osc}))

OSC_PRESENT=0, otherwise;

OSC=0, if OSC_PRESENT=0,

OSC=1, if OSC_PRESENT=1 and |C-A|<5a_{osc},

OSC=2, if OSC_PRESENT=1 and |C-A|>5a_{osc}.

3) CON=1,if
(A≥0 and A>B and B>C and (B>A(1-l_{min}T) or C>A(1-2l_{min}T)) or
(A<0 and A<B and B<C and (B<A(1-l_{min}T) or C<A(1-2l_{min}T)),

CON=0, otherwise.

4) DIV=1, if
(A≥0 and B>A and C>B) or
(A≤0 and B<A and C<B),

DIV=0, otherwise.

5) OFF=1, if

$A,B,C \in [e_{max}, e_{max}+a_{osc}]$ or
$A,B,C \in [-e_{max}-a_{osc}, -e_{max}]$,

OFF=0, otherwise.

The equations algorithmize the following principle. The system is accurate (ACC=0), if the transients are in the range $\pm e_{max}$. If there is a high-frequency oscillation superimposed to the transients, then OSC>0, if the amplitude of the high-frequency oscillation is above a_{osc} (is outside of the $\pm a_{osc}$ range around the transient), and even in this case we distinguish two cases (moderate and high). The transient converges (CON=0), if it stays inside the enveloping curve $e^{-?_{min}t}$. The transient diverges (DIV=1), if it is positive and monotonously increases, or it is negative and monotonously decreases (not self-correcting). In steady-state the transient shows offset error (OFF=1), if it is positive and long-lasting stays in the (a_{osc} wide) range given above the accuracy limit, or negative and continuously stays in the range given below the accuracy limit.

The inputs of the fuzzy expert are constituted by the outputs of the prefilter. In *Fig. 1.17* we summarize the base sets of the input signals, the different linguistic variables above them, and the membership functions of the fuzzy sets assigned to the linguistic variables.

Base sets	Fuzzy sets
X_1=OSC{0,1,2}	OKY,MOD,HIGH
X_2=CON{0,1}	OKY,NOK
X_3=DIV{0,1}	OKY,NOK
X_4=OFF{0,1}	OKY,NOK

OSC	0	1	2
OKY	1.0	0.2	0.1
MOD	0.2	1.0	0.2
HIGH	0.1	0.2	1.0

DIV	0	1
OKY	1.0	0.1
NOK	0.1	1.0

CON	0	1
OKY	1.0	0.2
NOK	0.2	1.0

OFF	0	1
OKY	1.0	0.2
NOK	0.2	1.0

Fig. 1.17 Linguistic input variables and memebership functions of the fuzzy expert

The fuzzy expert produces on its outputs the change (DP, DI, DD) of the parameters of the PID controller in comparison with the current values based on the rules of the fuzzy expert. The base sets of the outputs (changes) of the fuzzy expert, the linguistic variables above them and the membership functions of the fuzzy sets assigned to them are shown in *Fig. 1.18* (H=HIGH, L=LOW, NC=NO CHANGE).

Base sets	Fuzzy sets
Y_1=DP={-2,-1,0,1}	NH,NL,NC,PL
Y_2=DI={-1,0,2}	NL,NC,PH
Y_3=DD={-1,0,1,2}	NL,NC,PL,PH

DP	-2	-1	0	1
NH	1.0	0.2	0.1	0.0
NL	0.2	1.0	0.2	0.1
NC	0.1	0.2	1.0	0.2
PL	0.0	0.1	0.2	1.0

DD	-1	0	1	2
NL	1.0	0.2	0.1	0.0
NC	0.2	1.0	0.2	0.1
PL	0.1	0.2	1.0	0.2
PH	0.0	0.1	0.2	1.0

DI	-1	0	2
NL	1.0	0.2	0.0
NC	0.2	1.0	0.1
PH	0.0	0.1	1.0

Fig. 1.18 Linguistic output variables and membership functions of the fuzzy expert

Tuning the parameters of the PID controller a starting-point can be how a control engineering expert would modify the parameters of the controller based on his experience, if, observing the transients of the process, he detects oscillation, no convergence, divergence or offset error. We suppose that the expert would experiments with the process based on the following principle:

1. If there is oscillation, then the gain (K_P) should be decreased and the derivative time (T_D~K_D) should be increased.

2. If the transients are slowly decaying (no convergence), then the gain (K_P) and the derivative time (T_D~K_D) should be increased.

3. If the transients are diverging (the control is not self-correcting and exponential instability is to be expected), then the gain (K_P) should be decreased, the

derivative time (T_D~K_D) should be increased and the integrating time (T_I) should be increased (K_I~$1/T_I$ should be decreased).

4. If there is an offset error in steady-state, then the gain (K_P) should be increased and integration time (T_I) should be decreased (K_I~$1/T_I$ increases the effect of the integrator).

Therefore the following rules can be advised for tuning the parameters.

In case of oscillation:

if OSC=OKY *then* (DP=NC,DD=NC)
if OSC=MOD *then* (DP=NL,DD=PL)
if OSC=HIGH *then* (DP=NH,DD=PH)

In case of lack of convergence:

if CON=OKY *then* (DP=NC,DD=NC)
if CON=NOK *then* (DP=PL,DD=PL)

In case of divergence:

if DIV=OKY *then* (DP=NC,DI=NC,DD=NC)
if DIV=NOK *then* (DP=NL,DI=NL,DD=PH)

In case of offset:

if OFF=OKY *then* (DP=NC,DI=NC)
if OFF=NOK *then* (DP=PL,DI=PH)

The rules are evaluated separately. Since there are several outputs, the rules are considered separately for each output. The effect of the individual relations to the output is generated by the equations

$$\mathrm{R_i}\text{: } \textit{if } x \textit{ is } X^i \textit{ then } y \textit{ is } Y^i,$$

$$m_{Y_0^i}(y) := m_{D\circ R_i}(y) = \min\{m_{X^i}(x^*), m_{Y^i}(y)\},$$

$$m_{Y_0}(y) := m_{D\circ R}(y) = \max_i m_{Y_0^i}(y),$$

and is given in *Fig. 1.19*. After this the resultant effects $m_{Y_0}(y)$ of the relation groups (oscillation, convergence, divergence, offset) are generated. The membership functions of the outputs are shown in *Fig. 1.20*. The fuzzy outputs can be

defuzzified for each relation group based on two principles. The first principle applies weighted average (WAVG) based on the equation $y^*_{WAVG} = \mathrm{S}\, y_i m(y_i) / Y^{\#}$, where the cardinality $Y^{\#}$ is the number of the elements of the output base set (cardinality: 4 in case of DP and DD, 3 in case of DI). The second principle computes the center of gravity (COG) based on the equation $y^*_{COG} = \mathrm{S}\, y_i m(y_i) / \mathrm{S}\, m(y_i)$. The results are summarized in *Fig. 1.21*.

At the construction of the final fuzzy decision table (OSC, CONV, DIV, OFF) → (DP*, DI*, DD*) we modified the results: in case of OSC=CONV=DIV=OFF=0 (everything is OKY) we do not tune the controller parameters (DP=DI=DD=0). Otherwise only one output of the prefilter is effective (OSC≠0 or CONV≠0, or DIV≠0, or OFF≠0), and we take the *rounded* assigned fuzzified value (see *Fig. 1.22*).

Relation	DP				DI			DD			
OSC=OKY	-2	-1	0	1				-1	0	1	2
0	0.1	0.2	1.0	0.2				0.2	1.0	0.2	0.1
1	0.1	0.2	0.2	0.2				0.2	0.2	0.2	0.1
2	0.1	0.1	0.1	0.1				0.1	0.1	0.1	0.1
OSC=MOD	-2	-1	0	1				-1	0	1	2
0	0.2	0.2	0.2	0.1				0.1	0.2	0.2	0.2
1	0.2	1.0	0.2	0.1				0.1	0.2	1.0	0.2
2	0.2	0.2	0.2	0.1				0.1	0.2	0.2	0.2
OSC=HIGH	-2	-1	0	1				-1	0	1	2
0	0.1	0.1	0.1	0.0				0.0	0.1	0.1	0.1
1	0.2	0.2	0.1	0.0				0.0	0.1	0.2	0.2
2	1.0	0.2	0.1	0.0				0.0	0.1	0.2	1.0
CON=OKY	-2	-1	0	1				-1	0	1	2
0	0.1	0.2	1.0	0.2				0.2	1.0	0.2	0.1
1	0.1	0.2	0.2	0.2				0.2	0.2	0.2	0.1
CON=NOK	-2	-1	0	1				-1	0	1	2
0	0.0	0.1	0.2	0.2				0.1	0.2	0.2	0.2
1	0.0	0.1	0.2	1.0				0.1	0.2	1.0	0.2
DIV=OKY	-2	-1	0	1	-1	0	2	-1	0	1	2
0	0.1	0.2	1.0	0.2	0.2	1.0	0.1	0.2	1.0	0.2	0.1
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
DIV=NOK	-2	-1	0	1	-1	0	2	-1	0	1	2
0	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.1	0.1	0.1
1	0.2	1.0	0.2	0.1	1.0	0.2	0.0	0.0	0.1	0.2	1.0
OFF=OKY	-2	-1	0	1	-1	0	2				
0	0.1	0.2	1.0	0.2	0.2	1.0	0.1				
1	0.1	0.2	0.2	0.2	0.2	0.2	0.1				
OFF=NOK	-2	-1	0	1	-1	0	2				
0	0.0	0.1	0.2	0.2	0.0	0.1	0.2				
1	0.0	0.1	0.2	1.0	0.0	0.1	1.0				

Fig. 1.19 The effect of the individual relations to the output

Input	μ_{Yo}(DP)				μ_{Yo}(DI)			μ_{Yo}(DD)			
OSC	-2	-1	0	1				-1	0	1	2
0	0.2	0.2	1.0	0.2				0.2	1.0	0.2	0.2
1	0.2	1.0	0.2	0.2				0.2	0.2	1.0	0.2
2	1.0	0.2	0.2	0.1				0.1	0.2	0.2	1.0
CON	-2	-1	0	1				-1	0	1	2
0	0.1	0.2	1.0	0.2				0.2	1.0	0.2	0.2
1	0.1	0.2	0.2	1.0				0.2	0.2	1.0	0.2
DIV	-2	-1	0	1	-1	0	2	-1	0	1	2
0	0.1	0.2	1.0	0.2	0.2	1.0	0.1	0.2	1.0	0.2	0.1
1	0.2	1.0	0.2	0.1	1.0	0.2	0.1	0.1	0.1	0.2	1.0
OFF	-2	-1	0	1	-1	0	2				
0	0.1	0.2	1.0	0.2	0.2	1.0	0.2				
1	0.1	0.2	0.2	1.0	0.2	0.2	1.0				

Fig. 1.20 The effect of relation groups to the output

Input	Weighted average			Center of gravity		
OSC	DP^*		DD^*	DP^*		DD^*
0	-0.1		0.1	-0.25		0.25
1	-0.3		0.3	-0.75		0.75
2	-0.525		0.525	-1.4		1.4
CON	DP^*		DD^*	DP^*		DD^*
0	-0.05		0.1	-0.13		0.25
1	0.15		0.3	0.4		0.75
DIV	DP^*	DI^*	DD^*	DP^*	DI^*	DD^*
0	-0.05	0	0.05	-0.13	0	0.13
1	-0.325	-0.26	0.5	-0.86	-0.61	1.43
OFF	DP^*	DI^*	DD^*	DP^*	DI^*	
0	-0.05	0.06		-0.13	0.14	
1	0.15	0.6		0.4	1.28	

Fig. 1.21 Defuzzification by relation groups

Input	WAVG defuzzification			COG defuzzification		
	DP^*	DI^*	DD^*	DP^*	DI^*	DD^*
∀0(∀OKY)	0	0	0	0	0	0
OSC=1 (MOD)	-0.3	0.0	0.3	-0.75	0.0	0.75
OSC=2 (HIGH)	-0.5	0.0	0.5	-1.4	0.0	1.4
CON=1 (NOK)	0.15	0.0	0.3	0.4	0.0	0.75
DIV=1 (NOK)	-0.3	-0.3	0.5	-0.9	-0.6	1.4
OFF=1 (NOK)	0.15	0.6	0.0	0.4	1.3	0.0

Fig. 1.22 The final fuzzy decision table of the fuzzy expert

The denormalization can be made based on a uniform principle for each defuzzified controller parameter. Let p denote the controller parameter and let Δp denote the defuzzified parameter change according to the final fuzzy decision table of the fuzzy expert. Let the allowed interval of the control parameter be p_{min} ... p_{max}. Let the sensitivity of tuning be p_{sen}. Then the rule of denormalization is:

$$p_{new}=p_{old}+\Delta p(p_{max}-p_{min})/p_{sen}.$$

The value p_{new} computed in this way can be corrected to not exceed the parameter limits (saturation in case $p_{new}<p_{min}$ at $p_{min,}$ and in case $p_{new}>p_{max}$ at p_{max}).

1.8 Fuzzy Logic Toolbox in MATLAB environment

The Fuzzy Logic Toolbox [42] of MATLAB [39] allows the definition and use of fuzzy systems for modeling or control. The toolbox consists of the following components:

]

- FIS (Fuzzy Inference System) Editor
- Membership Function Editor
- Rule Editor
- Rule Viewer
- Surface Viewer

1.8.1 FIS Editor

The central menu is FIS, the other menus can be reached from it. When creating a new fuzzy inference system from scratch, the place to start is the FIS Editor. To do that, `fuzzy` should be typed. The FIS Editor displays a manu bar, which allows to open related GUI tools, open and save items, and so on (see File, Edit, View). Five pop-up menus are provided to change the functionality of the five basic steps (methods) in the fuzzy implication process (AND, OR, IMPLICATION, AGGREGATION, DEFUZZIFICATION).

Two types of fuzzy systems are allowed: `mamdani` or `sugeno` (default values are underlined). In the antecedents of the relations the linguistic variables can be connected by AND or OR operators. An extra `weight`$\in$ [0,1] can be defined for each relation which has an influence on the firing weight of the relation. The AND method (*T*-norm) may be `min` or `prod`, and the OR method (*S*-norm) may be `max` or `probor` in the antecedents. For performing the implication for a single relation we can choose as IMPLICATION method `min` or `prod` for `mamdani` type systems, but for `sugeno` type systems only `prod` is possible. The toolbox assumes individual-rule based inference. The AGGREGATION method of the results of the single relations may be `max`, `probor` or (bounded) `sum` for `mamdani` type systems, but for `sugeno` type systems only `singletons` are possible. The

DEFUZZIFICATION method for `mamdani` type systems may be `centroid`, `bisector`, `mom`, `lom` or `som` (middle, largest, smallest of maximum), but for `sugeno` fuzzy systems only `wtaver` or `wtsum` (weighted average, weighted sum) is possible.

1.8.2 Membership Function Editor

The Membership Function Editor is the tool that lets us display and edit all of the membership functions for the entire system, including both input and output variables.

There are no membership functions to start off with. On the left size of the graph area is a "Variable Palette" that lets us select the current variable. The membership functions from the current variable are displayed in the main graph. Below the Variable Palette is some information about the type and name of the current variable. There is one text field that lets us change the limits of the current variable's range (universe of discourse) and another that lets us set the limits of the current plot (which has no real effect on the system). In the lower right of the window are the controls that lets us change the name, position, and shape of the currently selected membership function. The membership functions can be chosen from following list:

- `sigmf`$=1/(1+\exp(-a(x-c)))$
- `gaussmf`$=\exp(-(x-c)^2/(2s^2))$
- `gbellmf`$=1/(1+|(x-c)/a|^{2b})$
- `trapmf` trapezoidal membership function with parameters a,b,c,d
- `trimf` triangular membership function with parameters a,b,c
- `zmf` *Z*-shape curved membership function with parameters a,b

During the definition of the membership functions all of the necessary number of parameters should be set.

1.8.3 Rule Editor

The Rule Editor contains a large editable text field for displaying and editing rules. The rules can be written in this field. The extra weight for the rule can be given at the end of the rule-line. After pressing Ctrl-Return the Rule Editor tries to parse every rule. Any rules that confuse the parser are marked with the # symbol and can be corrected.

Rules in the text field can be given in either of three formats (verbose form, symbolic form or indexed form), from which the verbose form is equivalent to our usual notation.

The indexed form is the version that the machine deals with. The first column-group beleongs to the input variables in the antecedents, the second column-group

belongs to the output variables in the consequents, the third column group contains the extra weight (listed between parentheses) and the fourth column group (listed after by :) indicates whether this is an AND (1) or OR (2) rule. The numbers in the first two column groups refer to the index number of the membership function, while the index of the column in the group identify the index of the input or output variable, respectively. Zero index number of the membership functions denotes that the corresponding variable does not take part (is neglected) in the relation.

1.8.4 Rule Viewer

The Rule Viewer displays a roadmap of the whole fuzzy inference process. Each rule is a row of plots, and each colum is a variable. The colums (left to right) represents the variables in the antecedent and cosequent, respectively. There are yellow index lines across the input variable plots that we can move left and right clicking and dragging with the mouse. This changes the input value and releasing the line, a new calculation is performed and we can see the result of the inference process. The resultant aggregate plots are shown in the last row. The defuzzified output values are shown by thick lines passing through the aggregate fuzzy sets.

The Rule Viewer shows one calculation at a time in great detail. In this sense, it presents a sort of micro view of the fuzzy inference system. Since it plots every part of every rule, it can become unwiedly for particularly large systems, but in general it performs well (depending how much screen place we devote it) with up to 30 rules and as many as 6 or 7 variables.

1.8.5 Surface Viewer

If we want to see the the entire output surface of the fuzzy system, that is the entire span of the output set based on the entire span of the input set, we need to use the Surface Viewer. For one-input and one-output fuzzy systems we can see the entire mapping in a 2D plot. Two-input and one-output systems generate 3D plots that MATLAB can adeptly manage.

The Reference Input field is used in situations when there are more inputs required by the system than are currently being varied. Suppose we have a four-input one-output fuzzy system and would like to see the output surface. The Surface Viewer can generate a 3D output surface where any two of the inputs vary, but the remaining inputs must be held constant since our monitors cannot display more than three-dimensional shape. In such case the Reference Input is a four element vector with NaN's holding the place of the varying two inputs while numerical values would indicate those values that remain fixed.

1.8.6 Data structure of saved fuzzy systems

Fuzzy systems defined by the FIS Editor can be saved in the form of a FIS matrix and there exist functions in the Fuzzy Logic Toolbox which may use the information provided in the FIS matrix. Simulink [40] allows the use of FIS matrix blocks in complex systems, especially as FLC (Fuzzy Logic Controller) in control systems.

The FIS matrix consists of the following components:

1. Name (name of the fuzzy system)
2. Type (type of the fuzzy system)
3. Inputs/Outputs (number of inputs and outputs)
4. NumInputMFs (number of input membeship functions for every input variable)
5. NumOutputMFs (number of output membeship functions for every output variable)
6. NumRules (number of rules)
7. AndMethod
8. OrMethod
9. ImpMethod
10. AggMethod
11. DefuzzMethod
12. InLabels (labels of the input variables)
13. OutLabels (labels of the output variables)
14. InRange ([low,high] ranges of the input variables in the order of input variables)
15. OutRange ([low,high] ranges of the output variables in the order of output variables)
16. InMFLabels (labels of the input mebership functions)
17. OutMFLabels (labels of the output mebership functions)
18. InMFTypes (types of the input membership functions)
19. OutMFTypes (types of the output membership functions)
20. InMFParams (parameters *a,b,c,d* of the input membership functions)
21. OutMFParams (parameters of the output membership functions)
22. RuleList (in indexed format)

The RuleList (in the higher versions of MATLAB) is divided into the parts Rule Antecedent (input group), Rule Consequent (output group), Rule Weight and Rule Connection (AND or OR in the antecedents), see the indexed format of the rules.

For *Sugeno fuzzy systems* OutMFParams contains the coefficients of the constant or linear deterministic output functions ordered in 'lines'. The lines are numbered and referred in the rule list as 'line1', 'line2', etc. The Rule Consequent (output group) refers to the line index in the appropriate rule.

All the parameters should be defined for a fuzzy system by using the FIS Editor, Membership Function Editor and Rule Editor.

2. FUNDAMENTALS OF GENETIC ALGORITHMS

Genetic algorithms give an alternative method to determine the global optimum for complicated problems having several local optima. To determine the optimum they use stochastic search procedures borrowed from natural evolution.

2.1 Base concepts of natural genetics

The interest towards the application of genetic algorithms has been significantly grown in the recent years. Comparing to the traditional search and optimization methods, genetic algorithms are robust, global and can be applied at a small cost, especially when there is no much information available about the examined system. Since genetic algorithms do not require information about the gradient of the objective function to be minimized, the applied search method, arising from its stochastic nature, is capable of searching the whole solution space and finding the optimum with great probability.

The genetic algorithm (GA) is a global stochastic search method, which learned its ideas by close observation of the natural biological evolution [47]. Genetic algorithms operate on the population of potential solutions, and apply the principle which states that by the survive of the ones most capable of living, (hopefully) better and better individuals come into being, that is, the better approximation of the solution is achieved. For every generation a new set of approximations is determined based on their suitability level (fitness) in the problem space in such a way that the approximations (individuals) are paired using operators taken from natural genetics. The process leads to a new population of individuals which produce better objective function values (adjust better to the environment), rather those individuals from which they were created, similarly the natural adaptation. In what follows, first the genetic base concepts, which form the base of genetic algorithms, are summarized.

- Gene: Material carrying genetic information, more or less permanent organization unit of the dezoxiribonucleinacid (DNA), encircled section of the polynucleotid double spiral, which alone or cooperating with other genes inside the boundaries specified by environmental conditions determines the appearance of certain properties or property groups. Managing to get unchanged in every successor cell or successor organism by repeated self-duplication, implements genetic continuity in the cell generations and generation series following each other. One gene consists of great number of nucleotides.
- Chromosome: Carrier of the inheriting material, the genes, which is organized into longer or shorter spring-like bodies. Chromosomes are embedded in the

plasma of the cell center. They keep their individual properties throughout the whole life cycle of the cell, but their material concentrates only at the fission of the cell as much as they are visible for microscopic observation. During observation the chromosomes are declassified, sorted into a plane, painted, taken a photo or transformed into a digitized picture. The separated and sorted picture is the chromosome set (chariogram), peculiar to the creature, in which the chromosomes are ranked according to their shape. Chromosomes, regarding their chemical compound, can contain DNAs, histons, non-histon-tempered proteins and metallic ions. The chromosomes, according to their morphological properties, can be more or less ranked into groups consisting of pairs (chariogram). Between shape-identically constructed chromosome pairs, in a certain life period of the cell (in the beginning of the meiosis), an odd attraction occurs. Individual members of the chromosome pairs fit lengthways, and single or multiple crossover happens. The paired chromosomes later separate again. The originated chromosomes usually resemble the ones made connection.

- Allel: One of the structurally and functionally altered versions of a gene. Different alternative allels of the same gene come into existence by mutation from each other, and are formed into each other the same way. A gene is represented by one of its allels at a place in the chromosome. In natural populations the most frequent form is usually called original or wild type. A mutant allel arisen by altering some point of the gene, compared to the wild allel, can be of variant strength: unable to display visible effect, displaying weaker or stronger visible effect than the wild type, opposite effect to the wild or qualitatively different effect from the wild type.
- Genotype: Sum of the genetic information stored in the chromosomal genes of the organism, which in interaction with the plasmon and environmental conditions determines the outer appearance (phenotype) of the organism.
- Phenotype: Sum of the perceptible, determinable (describable and measurable) inner and outer properties, resultant of the inheritable base and the life conditions.
- Individual: Organizational, physiological and reproduction-biological (genetic) unit of the living world, in other names living being or organism. Characteristic property of the individual is that it separates from its environment, exercises metabolism, propagates. Sum of individuals of identical origin, arisen via the genital process, is the population, while individuals arisen via non-genital way can not be considered as independent individuals until they are connected with their parent. Individuals in the nature usually make communities (supraindividual organizations) and fill the biosphere forming populations, associations and ecological systems.
- Population: Organizational unit above the individual level, which means sum of individuals sharing the same quality of a certain peculiarity, feature within the

species. According to the genetic interpretation population is a smaller or larger group of living beings belonging to the same species, in which the possibility of propagation between individuals exists. Every species exists in a form of populations. Due to the adaptation to the environment, the populations forming species more or less differ from each other regarding their phenotype and genotype. The ideal population consists of a large number of individuals; the number of individuals remains constant through generations, because it is not subject to the natural selection caused by the environment.

- Species: Fundamental unit of the systematization of the living beings and the evolution, which usually consists of populations of living beings similar to each other. By permanent or occasional exchange of the genes between the individuals of these populations, continuous variation of several properties arise. The emphasis of the biological species concept falls to the possibility of gene exchange. According to this the species is an advanced level possibility for propagation, whose members can exchange genes, and are separated from similar propagation communities by reproductive isolation.
- Breed: Fundamental economical or taxonomic (zoological or botanical) unit within species of cultivated plants and domestic animals. It is a group of individuals of a species, which is unambiguously distinguished by certain morphological and physiological properties.
- Evolution: Production of living material from lifeless (biogenesis) and development of diversity. Evolution is unequal in space and time, continuous and irreversible process. During evolution organism arise, which can balance disadvantageous environmental effects. Evolution genetics deals with examination of mechanics and factors of species arising and metamorphosis. Its fields are mutation, selection, isolation, recombination.
- Selection: Naturally or artificially initiated biological process, which hinders the survival or propagation of certain individuals or groups, while doesn't hinder others. In accordance with population genetics it means the non-random propagation of different phenotypes. Mutation is able to change only the composition of those groups in which genotype variance exists. Selection coefficient and selection pressure plays important role in the characterization of the mutation. The selection coefficient (s) is measurable at selection for qualitative character as the difference between the phenotype medium values (Ms) of the base population before the selection (M) and the descendant population: $s = M - Ms$. This difference, depending on the value of h^2 (inheritance rate), is inherited only to a certain extent in the descendant generation. Selection pressure is the intensity of the change of the gene frequency from generation to generation as the effect of selection. Selection is much more effective in large population, while smaller populations are dominated rather by genetic drift.

- Recombination (crossover): Exchange between genetic materials containing different genotypes, whose result is the recombined genotype of the descendant, differing from both parents; development of one or more new combination of genes from two, partly different parental genesets. Within the cell there is always a chance that partly different, but homogeneous genetic materials (chromosomes, DNA molecules) get so close that exchange can occur. The difference between identical gene positions (allels) can be smaller (point mutation) or larger (gene mutation). Recombination is the base of constructing the genetic map, because the frequencies can be transformed into relative distances, and with this the gene positions and connections can be computed. The gene order within a chromosome, the cistons within a gene and in these the distance between gene positions can be determined with the analysis of recombinants. For exchange between chromosomes two types of reaction models, break-and-fusion and pattern selection are known. The process occurs on the DNA level by means of working of several enzymes.
- Mutation: Change in succession materials, which is not a result of genetic recombination. A mutant is an individual in which by means of mutation at least one gene locus has changed. The mutant can be a new phenotype, but can be a cell component change too, invisible from outside. A characteristic property of mutants is that in morphological nature (height, number of leaves, spike shape) can be well distinguished. Biochemical mutants lost that property of the wild type which can syntethize physiologically important chemical compounds (vitamins, amino-acids). Depending on the level of change of the inheritable material, one can differ genomutation (number of chromosomes changes), chromosome mutation (genetic change leading to a new allel, which doesn't change the structure of the chromosome), plasmon mutation (change of plasmatic genetic components), plastidom mutation (extrachromosomal inheritance: inheritable change of plastids). Spontaneous mutations (arisen without use of mutagenes) and induced mutations (by effect of physical/chemical factors) can be differed. The unit of mutation is the muton, the smallest part of the DNA molecule, whose change can result in development of a mutant organism. This is usually a nucleotid pair (ciston, rekon) of the molecule. The absence, excess or change of the pending base pair transforms the triplet code and with this the information too, and comes down by means of replication. Mutations have important role in development of new species. The rate of spontaneous gene mutations is extremely small.
- Migration: Spreading of plant and animal species or their totality, flora and fauna. Gene migration is the transmission of genetic information from one population (emigration) to the other population (immigration) by migrant individuals or groups (gene drift).
- Fitness (competence value): Upon given external conditions the chance of an individual or individuals, incarnating a given genotype, for staying alive and giving

birth such viable descendants which will take part in forming the different generations. The ratio, with which the individuals with the pending genotype take part in forming the next generations, has been found really appropriate for expressing its numerical value. As base of comparison for determining the numerical value of the fitness, the mean of the population or the values characterizing the other genotypes of the population are used.

Genetic algorithms use ideas and rules taken from natural genetics with significant simplifications in the global stochastic optimum searching procedure. They primarily employ the following principles:

- Let there be a population of individuals. Every individual is a string over an alphabet. The individuals are different.
- Let there be genetic operations, which alter the individuals.
- Let there be a function, which defines a fitness (competence) value for every individual.
- After multiple changes the rearrangement of the population occurs based on the fitness value (reproduction).

The reproduction, which leads to the survival of chromosomes having high fitness value and dying out of those having smaller value, finally acts so that from generation to generation the chromosomes, from the point of view of the problem to be solved, become better.

Recently three schools of evolutionary algorithms can be differed: genetic algorithms, evolutionary strategies and genetic programming. Genetic algorithms were originally developed by J.H.Holland and D.E.Goldberg in the USA [11],[12]. Evolutionary strategies employ a different approach, its founders are I.Rechenberg and H.P. Schwefel in Germany. However, both schools are based on the ideas of evolution. Genetic programming generalizes these methods. If, namely, the original parameter-dependent system which should be optimized as a function of these parameters, is substituted with a theoretical construction such as a computational rule or a computer program, then the rule or program which optimally solves the formulated problem can be searched for. The possibility for connection is given by the fact that arithmetical or Boole-type expressions can be encoded with chromosomes. In case of programs the same goes for forks and recursions, where their prefix representation can be encoded with chromosomes. In case of genetic programming various fitness functions are used, which are based on errors occurring during use. Among these the most simple is the raw fitness. According to it if the j^{th} chromosome executes the i^{th} task with $E[i, j]$ value (e.g. this will be the value of a Boole-type expression as its effect) and the expected value is $f[i]$, then the fitness is $r[j] = \sum \left| E[i, j] - f[i] \right|$. Detailed treatment of genetic algorithms can be found in [15],[16],[47].

2.2 Principles of genetic algorithms

Although genetic algorithms can be used in case of multi-criterion (multiobjective) optimization problems, hereinafter only the optimization of single objective function is considered. A scalar criterion optimum problem can be always rephrased to a minimum problem. Usually several local minima are possible, but the problem is the calculation of the global minimum. Constraints can be taken into account as penalty functions embedded to the objective function. The problem is, consequently, the calculation of the global minimum of the unconstrained real function $f(x_1,...,x_{N\,\mathrm{var}})$:

$$\min f(x_1,...,x_{N\,\mathrm{var}}).$$

It is assumed that for every x_i variable the finite smallest and largest value is known, between which it can change. The x_i variables are typically real numbers $(x_i \in R^1)$, but exceptionally it is allowable that they take only integer values $(x_i \in Z^1)$. If x_i is real, it can be represented as a floating point number, but in case of predefined precision (number of bits PRECI), and lower and upper bounds, it can be represented as a binary combination in two's complement or Gray code. If x_i can be integer only, then a suitable base can be selected, which spans its domain and where an integer value codes it. In this sense real, binary and integer problems can be distinguished, where there is a significant similarity between the handling of the latter two. Hereinafter only the real and binary case is addressed.

The possible combinations of the $x_1,...,x_{N\,\mathrm{var}}$ natural variables are the individuals. The real form is the phenotype form, the coded form is the genotype form or chromosome. In the combination, in the place of the combination representing the x_i variable stands the tangible allel of the gene. For the sake of uniformity, the individuals represented directly as the real combination of $x_1,...,x_{N\,\mathrm{var}}$, are considered also as chromosomes (in this case, the phenotype is identical to the genotype). Certain number of individuals (N_{ind}) can be dynamically stored within the algorithm in coded (chromosome) form, which make the population. Individuals must be selected for recombination. The number of the selected individuals is defined by GGAP (generation gap) in such a manner that the number of individuals selected for crossover is the number of individuals of the population multiplied by this factor $(N_{ind} * GGAP)$.

For determining the place of the optimum, a global stochastic search method is used, which is based on the principles of selection, recombination, reinsertion, mutation and migration. The steps are realized by stochastic (random number generator-based) algorithms. At the reinsertion of the individuals created by recombination,

certain number of the best individuals prior to selection can be kept unchanged (elitist strategy). This number is indirectly determined by the reinsertion rate.

Genetic algorithms using one population (simple genetic algorithm, SGA) and using several sub-populations (multi-population genetic algorithm, MPGA), are distinguished. In case of multi-population genetic algorithm, migration is possible between sub-populations.

MATLAB can be effectively used for implementation of genetic algorithms [13],[14]. It is practical to implement multi-population genetic algorithms so that functions realizing selection, recombination, reinsertion and mutation remain the same as in the case of a single population. In this chapter we make acquainted with the concept of the GA toolbox of *Chipperfield et. al.* [13].

Objective functions should be of unified structure and exchangeable. In case of binary coding, before calculating the actual value of the objective function the value of the genotype (chromosome) value of the coded variables must be converted to phenotype (real) value, for which a conversion routine (binary string to real value, *bs2rv*) is needed. The real value of the objective function (*objfun*) before selection, reinsertion and migration is used to be converted to a fitness (competency) value transformed to a limited positive interval, thus during sophisticated algorithms the usually signed value of the objective function should not be taken into consideration. Different kinds of transformations are possible. For practical reasons it is expedient to limit the number of generations (MAXGEN), at which the algorithm stops to guard against the divergence of the global stochastic optimum search. It is practical to generate the starting population randomly (create binary population, *crtbp* and create real-valued population, *crtrp*).

For implementing the calculation of fitness, selection, recombination, reinsertion, mutation and migration it is practical to offer alternative possibilities. It is practical to automatically derive the parameters which influence running time and memory consumption from the dimension of the problem $\left(DIM = N_{\text{var}}\right)$.

Fig. 2.1 shows the theoretical structure of a simple (one population) binary genetic algorithm. In case of binary population *FielD* has as many columns as many variables exist, and for every variable it defines the accuracy (*len*), lower (*lb*) and upper (*ub*) bound, *coding* (1=Gray, 0=standard binary), *scale* (1=logarithmic resolution, 0=binary resolution), and whether the lower bound is part of the represented interval (in case of *lbin*=1 it is part of it, otherwise not), and whether the upper bound is part of the represented interval (in case of *ubin*=1 it is part of it, otherwise not). The *rep* function (which originally did not exist in the earlier versions of MATLAB) replicates the matrix given as first parameter in horizontal and vertical direction.

Naturally, the parameters of the multipopulation genetic algorithm are more ample. Different alternative functions can be selected for implementation of selection, recombination and mutation. The *FielD* consists only of upper and lower bounds.

```
NIND=40;
MAXGEN=300;
NVAR=20;
PRECI=20;
GGAP=0.9;

%Build field descriptor
FieldD=[rep([PRECI],[1,NVAR]);rep([-512;512],[1,NVAR]);... rep([1;0;1;1],[1,NVAR])];

%Initialize population
Chrom=crtbp(NIND,NVAR*PRECI);

gen=0;

%Evaluate initial population
ObjV=objfun1(bs2rv(Chrom,FieldD));

%Generational loop
while gen<MAXGEN,
        %Assign fitness values to entire population
        FitnV=ranking(ObjV);

        %Select individuals for breeding
        SelCh=select('sus',Chrom,FitnV,GGAP);

        %Recombine individuals (crossover)
        Selch=recombin('xovsp',Selch,0.7);

        %Apply mutation
        SelCh=mut(SelCh);

        %Evaluate offspring, call objective function
        ObjVSel=objfun1(bs2rv(SelCh,FieldD));

        %Reinsert offspring into population
        [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);

        %Increment counter
        gen=gen+1;

end
```

Fig. 2.1 Structure of a simple genetic algorithm (SGA)

```
%Define GA Parameters
GGAP=0.8;                       %Generation gap
XOVR=1;                         %Crossover rate
MUTR=1/NVAR;                    %Mutation rate
MAXGEN=1220;                    %Maximum no. of generations
INSR=0.9;                       %Insertion rate
SUBPOP=8;                       %No. of subpopulations
MIGR=0.2;                       %Migration rate
MIGGEN=20;                      %No. of generations/migration
NIND=20;                        %No. of individuals/subpop

%Specify other funtions as strings
SEL_F='sus';                    %Name of the selection funtion
XOV_F='recdis';                 %Name of the recombination funtion
MUT_F='mutbga';                 %Name of the mutation function
OBJ_F='objharv';                %Name of objective function

Chrom=crtrp(SUBPOP*NIND,FieldD);
gen=0;

ObjV=feval(OBJ_F,Chrom);

%Generational loop
while gen<MAXGEN,
        %Fitness assignment to whole population
        FitnV=ranking(ObjV,2,SUBPOP);

        %Select individuals from population
        SelCh=select(SEL_F,Chrom,FitnV,GGAP,SUBPOP);

        %Recombine selected individuals
        SelCh=recombin(XOV_F,SelCh,XOVR,SUBPOP);

        %Mutate offspring
        SelCh=mutate(MUT_F,SelCh,FieldD,[MUTR],SUBPOP);

        %Calculate objective function for offsprings
        ObjVOff=feval(OBJ_F,SelCh);

        %Insert best offspring replacing worst parents
        [Chrom,ObjV]=reins(Chrom,SelCh,SUBPOP,...
                           [1 INSR],ObjV,ObjVOff);

        %Increment counter
        gen=gen+1;

        %Migrate individuals between subpopulations
        if (rem(gen,MIGGEN)==0)
            [Chrom,ObjV]=migrate(Chrom,SUBPOP,[MIGR,1,1],ObjV);
        end
end
```

Fig. 2.2 Generation loop in case of multipopulation genetic algorithm (MPGA)

2.3 Fitness functions

The objective function (scalar-valued criterion) is a $\mathbf{x} = (x_1,...,x_{N_{\text{var}}}) \mapsto f(\mathbf{x})$ mapping. Values of the objective function can be positive and negative too. The values of the $F(\mathbf{x}) = g(f(\mathbf{x}))$ fitness function are expected to preserve order and be positive.

Let the individuals of the population in phenotype (real) form be the $\mathbf{x}_i$ vectors: $\{\mathbf{x}_i\}_{i=1}^{N_{ind}}$. With the help of the objective functions the $\{f(\mathbf{x}_i)\}_{i=1}^{N_{ind}}$ objective function values can be computed for the population and can be placed in the order of the individuals in the *ObjV* vector. The ranking function can order fitness (competency) values to the values of the objective function based on specified principles and place them in the *FitnV* vector.

1. In case of positive objective function the proportional fitness is defined by
$$F(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{i=1}^{N_{ind}} f(\mathbf{x}_i)}.$$
2. In case of scaling
$$F(\mathbf{x}_i) = af(\mathbf{x}_i) + b$$
where *a*, *b* are the scaling factor and offset, respectively. The scaling function automatically determines the values of *a* and *b* from the input variable S_{mult} so that if the average value of *ObjV* is f_{ave}, then the upper bound of the maximal fitness is $f_{ave} * S_{mult}$. If one of the values of *ObjV* is negative, then the scaling tries to find an appropriate *b* value so that the fitness becomes positive. Since it is not always possible, using of scaling should be avoided if $f(\mathbf{x})<0$.
3. Linear ranking can be used for positive and negative *ObjV* values too. In the first step the $f(\mathbf{x}_i)$ values are sorted so that in the $f(\mathbf{x}_i) \to pos$ mapping $\max f(\mathbf{x}_i) \to pos(f(\mathbf{x}_i)) = 1, \ldots, \min f(\mathbf{x}_i) \to pos(f(\mathbf{x}_i)) = N_{ind}$ is satisfied.

   After this, in case of linear ranking
$$F(\mathbf{x}_i) = 2 - sp + 2 * (sp - 1)\frac{pos(f(\mathbf{x}_i)) - 1}{N_{ind} - 1},$$
   that is, keeping the order, *2-sp* is assigned to the maximum of the objective function values and *sp* to the minimum (e.g. in case of *sp=2* the most competent individual gets the fitness value 2, and the least competent gets 0).

4. In case of nonlinear ranking, after the first step the root *X* of the polynom

$$(sp - N_{ind})x^{N_{ind}-1} + sp * x^{N_{ind}-2} + ... + sp * x + sp = 0$$

is determined and with it the fitness value

$$F(\mathbf{x}_i) = \frac{N_{ind} * X^{pos(f(\mathbf{x}_i))-1}}{\sum_{i=1}^{N_{ind}} X^{i-1}}$$

is calculated. This method assigns *sp* to the minimal objective function value while to the others it assigns nonlinearly descending values, keeping the order.

The reason of assigning the fitness to the ranking position instead of the value of the objective function is used to forbid early convergence to a local minimum.

2.4 Selection methods

Selection is the determination of the number of occasions of an individual is selected for recombination (crossover); that is, the number of occasions it takes part in making a descendant (offspring). Selection consists of two independent processes:

1) determination of occasions for an individual wich it can count on
2) converting the expected number of occasions to a discrete number of individuals.

The first part transforms the fitness into an expected value of a probability of an individual takes part in the reproduction. The second part is the random selection of individuals for reproduction, which is based on the relative competence value of the individuals. In judging the selection process, three parameters can help. The *bias* is the difference between the actual and expected selection probability. The *spread* is the value interval of the possible experiences, in which an individual can take part. Bias characterizes precision, while spread characterizes consistency. *Efficiency* is characterized by the time-complexity of the genetic algorithm. In choosing the selection method the goal is to provide zero bias, minimal spread and little time needs. In practice, roulette-wheel selection and stochastic universal sampling are the most used.

1. Roulette wheel selection

   A real valued interval *Sum* is defined, which can be the sum of the expected selection probability of the individuals, or the sum of the fitness values of the individuals in the population. The individuals can be mutually unambiguously projected to contiguous intervals within the interval [*0,Sum*]. Sizes of the intervals of the individuals correspond with their fitness values. For example, *Fig. 2.3* depicts six individuals, of which the 3. individual has the biggest fintness value, while 4. and 6. has the smallest. An individual can be selected by generating a uniformly distributed random number on the interval [*0,Sum*], and the individual whose

segment spans the generated number, is selected. The process is repeated until the necessary number of individuals is selected.

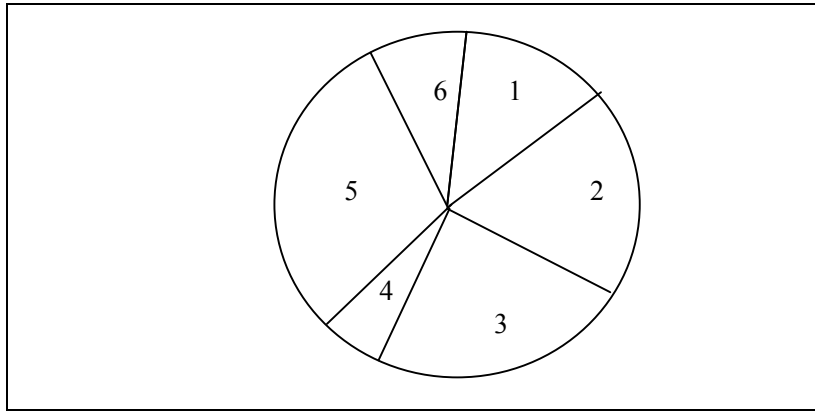


Fig. 2.3 Roulette-wheel selection method

The fundamental roulette-wheel selection method is the stochastic sampling with replacement (SSR), in which the sizes of the segments and the selection probability remains the same during the whole selection phase, and the selection of the individuals is done by the aforementioned method. Stochastic sampling with partial replacement (SSPR) expands the SSR method by altering the size of the segment of the selected individual. Each time an individual is selected, the value of its segment is reduced by 1 while it is positive, then sticks to zero. This provides an upper limit for spread, but the lower limit is zero, therefore the bias is bigger than in the SSR.

The remaining sampling methods can be divided into two different phases. In the integer phase the individuals are selected deterministically by the integer part of the expected value of the experiments. The remaining individuals are selected randomly by the fractional part of the expected value. Remainder stochastic sampling with replacement (RSSR) uses roulette-wheel selection for the individuals to be selected non-deterministically. During the roulette-wheel phase the fractional part remains unchanged, thus they stay in race. The RSSR provides zero bias and lower limit spread.

The upper bound is limited only by the number of samples selected by fractional part and size of the integer part of the individuals. Remainder stochastic sampling without replacement (RSSWR) substitutes the fractional part of the expected value of the individual with zero, if it was selected during the fraction

phase. This method results in minimal spread, but causes bias, because it prefers smaller fractional parts.

2. Stochastic universal sampling (SUS) is a one-phase sampling with minimal spread and zero bias. Unlike the roulette-wheel selection, when one selection pointer was used, the SUS method uses N equidistant pointers, where N is the number of selections. The population is randomly shuffled and a random number is generated in the [*0,Sum/N*] interval, which becomes the pointer *ptr*. The N individuals will be marked by $\{ptr, ptr + \frac{sum}{N}, \ldots, ptr + (N-1)\frac{sum}{N}\}$. Pointers spanned by the corresponding fitness values mark the selected individuals.

Roulette-wheel methods are of $s(N \log N)$ complexity while the SUS method is of $s(N)$.

2.5 Recombination (crossover)

Method of generating new chromosomes (individuals), in case of genetic algorithm, is the recombination (crossover). Different methods of implementing the recombination are used for binary and real populations. Recombination is preceded by selection. Selected individuals are paired and the crossover happens in the pair. From the individuals generated by the recombination those are picked (randomly or by considering their fitness values) which will be reinserted to the population.

1. Recombination in case of binary population
   In case of a binary population one-point or multipoint recombination is used. Let L be the length of the binary chromosome string.

   In case of one-point crossover for every chromosome pair a uniformly distributed random integer i is selected from the $[1, L-1]$ interval, and the genetic information is exchanged between the pair around the bit indexed by i, which results in two new individuals.

   In case of multipoint crossover M recombination points $k_i \in \{1,2,...,L-1\}$ are selected in a random manner without repetition for every selected odd/even chromosome pair. The k_i crossover places are sorted in ascending order, and the bits are exchanged between the adjacent crossover points between the parents, resulting in two new descendants. The bits between the first allel bit (1) and the first crossover point remain unchanged in the descendants too, see *Fig. 2.4*.

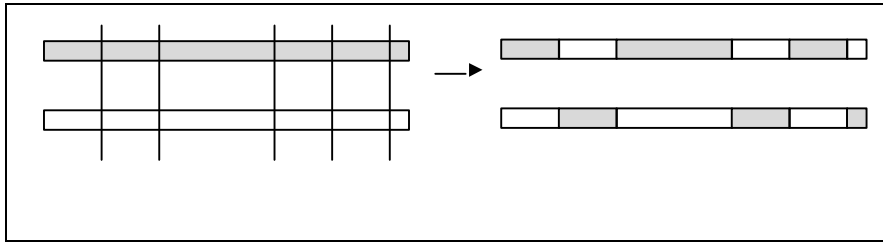


Fig. 2.4 Multipoint crossover (m=5)

Different variations of the recombination operators are based on the same experience: those parts of the chromosome, which affect the quality index (objective function value) significantly into good direction, usually follow each other in non-connecting substrings. Multipoint crossover prevents convergence towards a big fitness individual in the early stage of the algorithm, thus prohibits sticking to a local minimum. Thus crossover makes global optimum search robust.

Uniform crossover generalizes the previous scheme so that every chromosome place can become crossover point. This can be implemented by generating a mask of length L, which equals the length of the chromosome, for every odd/even chromosome pair. The bits of the parents are exchanged according to the mask. For example let P_1, P_2 be the parents, M the mask, O_1, O_2 the offsprings. The process is as follows:

$$
\begin{aligned}
P_1 &= 1\,0\,1\,1\,0\,0\,0\,1\,1\,1 \\
P_2 &= 0\,0\,0\,1\,1\,1\,1\,0\,0\,0 \\
M &= 0\,0\,1\,1\,0\,0\,1\,1\,0\,0 \\
O_1 &= 0\,0\,1\,1\,1\,1\,0\,1\,0\,0 \\
O_2 &= 1\,0\,0\,1\,0\,0\,1\,0\,1\,1
\end{aligned}
$$

The bit exchange can be controlled by a probability parameter, which can be used to supervise the rate of tearing the bits during the recombination without causing distortion in the direction of the representation length. It must be noted that in case of real allels the analogy of the uniform crossover is the discrete recombination.

From the special crossover operators two are emphasized. Shuffle operator uses one-point crossover, but before recombination shuffles the bits in both parents. After recombination the bits of the offsprings are unshuffled. Reduced surrogate operator enforces descendants altering from the parents. It tests the selected

crossover places, and accepts only if the genes of the parents differ at the crossover places.

2. Recombination in case of real population

In case of real populations the binary recombination operators are not applicable. The new phenotype values, in case of real population, arise from and between the phenotype values of the odd/even parents.

In case of intermediate recombination the phenotype values O_1 or O_2 of the offspring arise from the phenotype values of the parents P_1 and P_2 with respect to a factor **a**, which is a random vector. Its components are selected from the $[-0.25, 1.25]$ interval, $\dim \mathbf{a} = N_{\text{var}}$:

$$s = P_1 + \mathbf{a} .* (P_2 - P_1).$$

For even and odd offsprings different **a** vectors are selected. In case of intermediate recombination the offsprings arise in a slightly bigger hypercube as the one spanned by the parents, see *Fig. 2.5*. The size of the hypercube is limited by **a**.

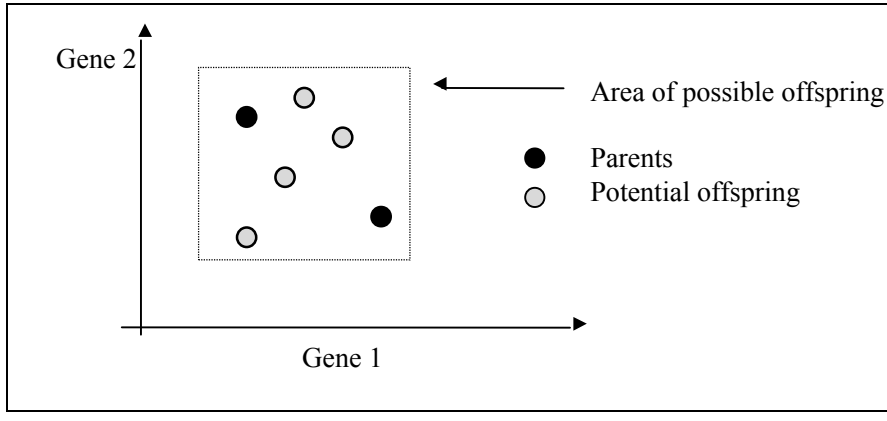


Fig. 2.5 Intermediate recombination

In case of line recombination the rule $s = P_1 + a * (P_2 - P_1)$ is used, in which α is a random number. This can be interpreted as a degenerating case of the intermediate recombination, where all components of the vector **a** are equal, don't depend on the phenotype variable index, see *Fig. 2.6.*

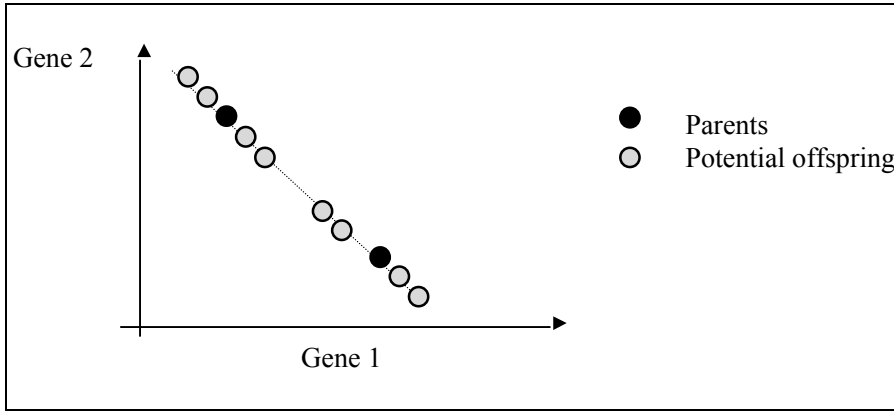


Fig. 2.6 Line recombination

It must be noted that applying binary recombination operators on real populations would be senseless, because the resulting genetic material in the recombination would significantly differ from the values of the decision variables (x_i) of the parents, which contradicts the expectation of the offspring containing genes from both parents. Intermediate and line recombination operators extinguish this limitation, because they directly affect the decision variables.

In the implementation of genetic algorithms it is practical to reach the special crossover functions via a high-level entry function (recombin). The name of the special crossover function is a parameter of the high-level entry function. In case of binary populations special crossover functions should be used for one-point, multipoint and mixed crossover operators (*xovsp, xovdp, sovsh*), and their reduced operator versions (*xovsprs, xovdprs, xovshrs*), which appropriately parameterized, can call a common multiport crossover function (*xovmp*), which interprets the parameters and implements the special binary crossover type in a uniform frame. Special recombination functions realizing discrete, intermediate, line operators (*recdis, recint, reclin*) applicable for real populations can be reached via the high-level entry function (*recombin*).

2.6 Mutation

In the natural evolution the mutation is a random process, in which an allel of the gene is substituted to create a new genetic structure. In the genetic algorithm the mutation is used randomly, at a low probability rate; its typical value is between 0.001 and 0.01. Usually it is used as a background operator to ensure that the search

probability of every string is not zero and the good genetic information is not lost during selection and crossover.

The effect of the mutation in case of a linear chromosome is illustrated with the example in *Fig. 2.7*. Let x be in the (0,10) interval, the coding can be standard binary or Gray code, and the mutation point let be the 3rd bit in the linear string.

Mutation point →											binary	Gray
Original string-	0	0	0	1	1	0	0	0	1	0	0.9659	0.6634
Mutated string-	0	0	1	1	1	0	0	0	1	0	2.2146	1.8439

Fig. 2.7 Effect of the mutation in different codes

It can be well understood that mutating a single bit (depending its position) can cause significant changes in the value of the phenotype in both coding. The example also provides a chance to clarify the application of the binary → real conversion. As known, the prescriptions of the conversion of a natural x_i variable (here only one exists) is defined by the [*len*, *lb*, *lu*, *code*, *scale*, *lbin*, *luin*]' column in FieldD during the conversion in Phen=*bs2rv*(Chrom, FieldD). Here:

[10,0,10,0,0,0,0] binary
[10,0,10,1,0,0,0] Gray

It is easy to make sure that if a standard binary code of an integer is $(d_N, d_{N-1}, ..., d_1)$, then its Gray code (in which the code of neighbouring numbers differs only at one bit) is $(g_N, g_{N-1}, ..., g_1)$, where

$$g_N = d_N,$$
$$g_i = d_i + d_{i+1} \pmod 2, \qquad i < N.$$

From the other side, the conversion from Gray to standard binary code is

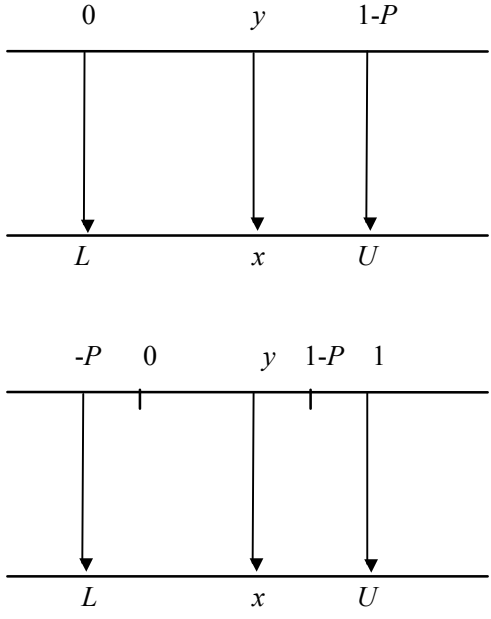
$$d_N = g_N$$
$$d_i = g_N + g_{N-1} + \cdots + g_i \pmod 2, \qquad i < N.$$

Gray code can be advantageously used in genetic algorithms, because the stochastic search is less likely to be deceived by the regular Hamming distance between the quantizing intervals.

Excluding the 0 and 10 upper and lower bounds needs further explanation. Let *n=len* be the bit number and let $P=\left(\frac{1}{2}\right)^n$ be the precision. Then the binary string $(b_1, b_2, ..., b_n)$ corresponds to the fixed point number

$$y=\sum_{i=1}^{n} b_i\left(\frac{1}{2}\right)^i .$$

Let us consider the two important cases: in the first, the lower (L) and upper (U) bounds are not excluded, and the other, when they are excluded (this is the case in the example above). The correspondence and conversion rule to the x phenotype value is depicted on *Fig. 2.8*.



$$x=\frac{y}{1-P}(U-L)+L$$

$$x=\frac{y+P}{1+P}(U-L)+L$$

Fig. 2.8 Computation of x, if L,U are included in the code (top), and if L,U are excluded from the code (bottom)

The other two cases (only the upper or lower bound is excluded) can be similarly discussed.

In case of real populations the perturbation of the value of the gene or the random selection of the new values can be used for implementing the mutation. It is practical to specify a mutation rate (*Mutrate*, e.g. $1/N_{\text{var}}$), a shrinking (*Mutshrink*) parameter, and an accuracy (*Accur*) parameter. Mutation is carried out by adding an

increment, supervised by a mask, to the original value of the variable. The mask (*Mutmask*) selects the variable x for mutation with *Mutrate* probability, and the modified variable is marked with equal probability for change in + or – direction (*Mutmask* is 0,+1, or -1). The increment is randomly selected according to $Delta = \sum_{i=1}^{m-1} a_i 2^{-i}$, where *m=Accur* is typically *20*, and the value of a_i becomes *1* with *1/m* probability, otherwise zero. The new value of the variable after the mutation is the following:

$$x_{mut} = x + Mutmask * \frac{U - L}{2} * Mutshrink * Delta$$

In case of $m = 20$ the selected mutation operator is able to find the place of the optimum with $0.5(U - L)Mutshrink \cdot 2^{-19}$ precision.

At the implementation of the genetic algorithm it is advised to reach the binary (*mut*) and the real (Breeder Genetic Algorithm, *mutbga*) functions through a high level entrance function (*mutate*).

2.7 Reinsertion

After the new generation was created from the old population by selection, recombination and mutation, the objective values of the individuals of the new population have to be determined. If less individuals were born during recombination, then the difference between the number of individuals in the new and old population is defined as generation gap. If only a few new individuals came into existence, it is called steady-state or incremental genetic algorithm. If the best individuals are deterministically allowed to live from generation to generation, it is called elitist strategy.

Selection of the individuals for reinsertion can be done in a random manner (uniform selection) or according to the fitness value of the individuals (fitness based selection). The reinsertion rate of the individuals can be given proportionally to the size of the population (to the size of the subpopulation in case of multipopulation), which indirectly determines the number of elite individuals.

In the genetic algorithm the random, fitness-based or elitist reinsertion can take place in a common function (*reins*). In case of fitness-based reinsertion the values of the objective function *ObjVSel* of the individuals must be previously determined.

2.8 Migration

In case of multipopulation the population consists of subpopulations. Handling of subpopulations with the single population functions of selection, recombination, mutation and reinsertion can be done with a simple loop, which can well utilize the

high-level entry functions. The new subpopulation will always be reinserted to the corresponding old subpopulation.

The exchange of individuals between subpopulations is called migration. For the migration, the migration rate must be given as a percentage of the size of the subpopulation, furthermore the type of the selection for migration (random or fitness-based), and the migration topology must be chosen. The number of migrating individuals between the subpopulations is always constant $\left(Migrate * N_{ind} / N_{subpop}\right)$. The selected individuals according to the specified topology penetrate one or more other subpopulation. The following topologies are called ring, neighbourhood, and unrestricted topologies, respectively, as shown in *Fig. 2.9-11*.

For implementing the migration it is practical to create a high-level function (*migrate*), whose inputs are the migration parameters and the type of topology.

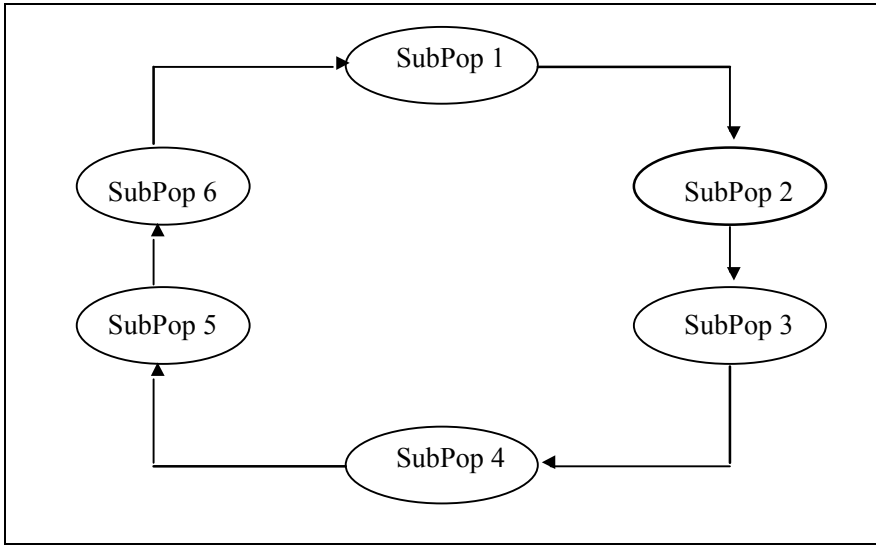


Fig. 2.9 Ring topology

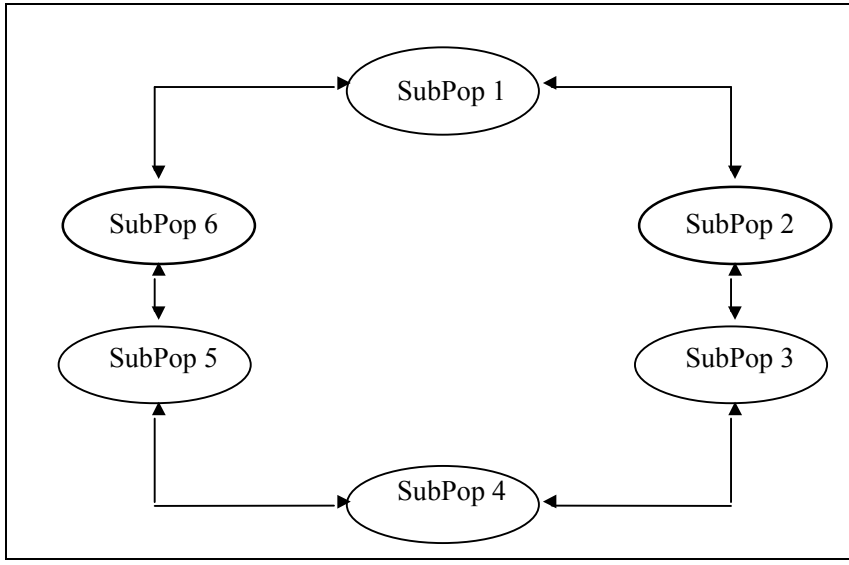


Fig. 2.10 Neighbourhood topology

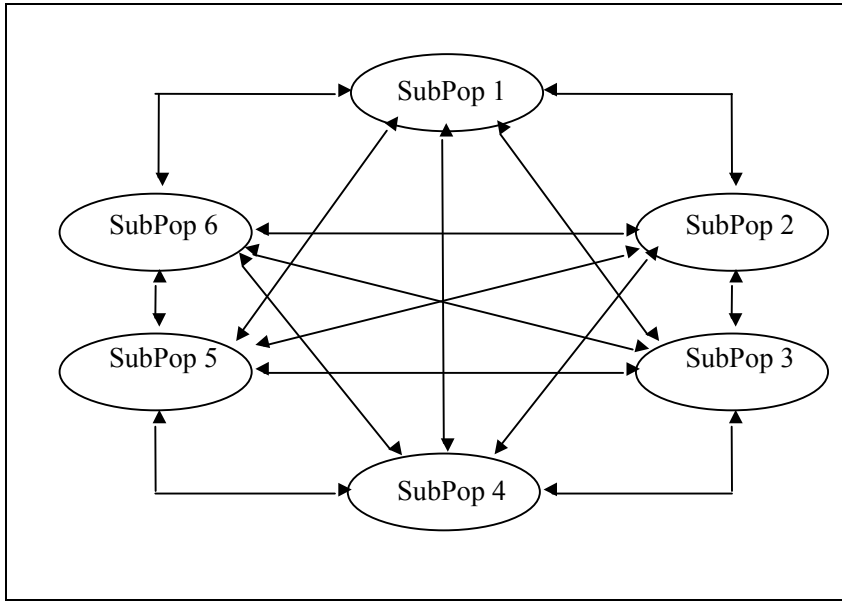


Fig. 2.11 Unrestricted topology

2.9 Testing genetic algorithms

To test the efficiency and convergence of genetic algorithms, the literature suggests various functions (*objfun*), usually with more than one local minimum. It is advised to implement a frame program for the single population (*sga*) and multipopulation (*mpga*) case, which provides sufficient graphical interface to test the running. In what follows a few characteristic functions and simple control problems are presented, which can be well used for testing the efficiency of genetic algorithms.

2.9.1 Typical test functions and their minimum places

- De Jong's function:

$$f_1(\mathbf{x}) = \sum_{i=1}^{N} x_i^2 , \qquad -512 \le x_i \le 512$$

  global minimum: $\forall x_i = 0$, $f_1(\mathbf{0}) = 0$.

- Axis-parallel hyperellipsoid:

$$f_{1a}(\mathbf{x}) = \sum_{i=1}^{N} i * x_i^2 , \qquad -512 \le x_i \le 512$$

  global minimum: $\forall x_i = 0$, $f_{1a}(\mathbf{0}) = 0$.

- Rotated hyperellipsoid:

$$f_{1b}(\mathbf{x}) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2 , \qquad -65 \le x_i \le 65$$

  global minimum: $\forall x_i = 0$, $f_{1b}(\mathbf{0}) = 0$.

- Rosenbrock's banana-shaped valley:

$$f_2(\mathbf{x}) = \sum_{i=1}^{N} 100 * (x_{i+1} - x_i)^2 + (1 - x_i)^2 , \qquad -2 \le x_i \le 2$$

  global minimum: $\forall x_i = 1$, $f_2(1,...,1) = 0$

- Rastrigin's function:

$$f_6(\mathbf{x}) = 10 * N + \sum_{i=1}^{N} \left(x_i^2 - 10 * \cos(2\pi x_i) \right), \qquad -5.12 \le x_i \le 5.12$$

  global minimum: $\forall x_i = 0$, $f_6(\mathbf{0}) = 0$.

- Schwefel's function:

$$f_7(\mathbf{x}) = -\sum_{i=1}^{N} \left(x_i * \sin\left(\sqrt{|x_i|} \right) \right), \qquad -500 \le x_i \le 500$$

  global minimum: $\forall x_i = 420.9687$, $f_7(420.9687,...,420.9687) = N * 418.9829$.

- Griewangk's function:

$$f_8(\mathbf{x}) = \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad -600 \le x_i \le 600$$

  global minimum: $\forall x_i = 0$, $f_8(\mathbf{0}) = 0$.

- Sum of powers having different exponents:

$$f_9(\mathbf{x}) = \sum_{i=1}^{N} |x_i|^{i+1}, \qquad -1 \le x_i \le 1$$

  global minimum: $\forall x_i = 0$, $f_9(\mathbf{0}) = 0$.

Analysis of the test functions reveals that the De Jong's function is a relatively simple, convex and unimodal function. The axis-parallel hyperellipsoid is similar to the De Jong's function, convex and unimodal. The rotated hyperellipsoid comes into existence from it by rotation and it is similarly convex and unimodal. The global optimum, in the case of Rosenbrock's banana function lies inside a long, narrow, parabola-shaped valley. It is relatively easy to find the valley, but the convergence to the global optimum is rather critical. Rostrigin's function is based on De Jong's function, which has been supplemented by a cosine modulation, which causes many local optima. The test function is multimodal to a great extent. The distribution of the local minima is uniform. Schwefel's function is characterized by that its global minimum place is geometrically far from the next best local minimum, thus the search algorithms can potentially converge to a wrong direction. Grienwangk's function is similar to Rastrigin's function, but the local minima are of uniform distribution. The sum of powers having different exponents is a frequently used unimodal test function.

2.9.2 Simple dynamic optimum control problems

- Control of a double integrator

  System:

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u$$

$$y = x_2$$

  Boundary conditions:

$$\begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \qquad \begin{pmatrix} x_1(1) \\ x_2(1) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

  Control:

$$u(t), \quad 0 \le t \le 1, \quad -15 \le u(t) \le 15$$

  Objective function:

$$\int_0^1 u^2(t)\,dt \to \min$$

- Harvest problem

  System:

$$x_{k+1} = ax_k + u_k \qquad (a=1.1)$$

  Boundary condition:

$$x_0 = x_N = 100$$

  Control:

$$(u_0, u_1, ..., u_{N-1}) \in R^N, \qquad 0 \le u_k \le 10N$$

  Objective function:

$$f(\mathbf{u}) = -\sum_{k=0}^{N-1} \sqrt{u(k)} \to \min$$

Table 2.1 Characteristic cases for the discrete time LQ problem

index	N	$x(0)$	s	r	q	a	b	exact solution
1	45	100	1	1	1	1	1	16180.3399
2	45	100	10	1	1	1	1	109160.7978
3	45	100	100	1	1	1	1	10009990.0200
4	45	100	1	10	1	1	1	37015.6212
5	45	100	1	1000	1	1	1	287569.3725
6	45	100	1	1	0	1	1	16180.3399
7	45	100	1	1	1000	1	1	16180.3399
8	45	100	1	1	1	0.01	1	10000.5000
9	45	100	1	1	1	1	0.01	431004.0987
10	45	100	1	1	1	1	100	10000.9999

- Discrete time LQ problem

  System:

$$x_{k+1} = ax_k + bu_k$$

  Initial value:

$$x_0 = 100$$

  Control:

$$(u_0, u_1, ..., u_{N-1}) \in R^N$$

$$-100 \le u_0 \le 20$$
$$-70 \le u_1 \le 20$$
$$-50 \le u_2 \le 20$$
$$-30 \le u_k \le 20,\ k = 3,4,...,N-1$$

Objective function:

$$f(\mathbf{x},\mathbf{u}) = q \cdot x_N^2 + \sum_{k=0}^{N-1} \left\{ s \cdot x_k^2 + r \cdot u_k^2 \right\} \to \min$$

- Continuous time LQ problem:

System:

$$\frac{dx}{dt} = x + u$$
$$y = x$$

Initial condition:

$$x(0) = 100$$

Control:

$$u(t), \qquad 0 \le t \le T, \qquad -600 \le u(t) \le 0$$

Objective function:

$$f(\mathbf{x},\mathbf{u}) = c_E x^2(T) + \int_0^T \left\{ c_x x^2(t) + c_u u^2(t) \right\} dt \to \min$$

- Push cart problem

System:

$$\begin{pmatrix} x_{1,k+1} \\ x_{2,,k+1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix} \begin{pmatrix} x_{1,k} \\ x_{2,,k} \end{pmatrix} + \begin{pmatrix} 0 \\ 1/N^2 \end{pmatrix} u_k$$

Initial condition:

$$\begin{pmatrix} x_{1,0} & x_{2,0} \end{pmatrix}^T = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$$

Control:

$$(u_0, u_1, ..., u_{N-1}) \in R^N, \qquad 0 \le u_k \le 5$$

Objective function:

$$f(\mathbf{x},\mathbf{u}) = -x_{1,N} - \frac{1}{2N} \sum_{k=0}^{N-1} u_k^2 \to \min$$

The next table summarizes the names of functions and the frame program in which it is examinable. When the dynamic optimization problem contains final value condition, the objective function is supplemented by a punishing function, which punishes the final value error. The scaling of the original objective function and the punishing function must be selected.

Table 2.2 Typical test functions

Function	SGA	MPGA	objfun
De Jong	+	+	objfun1
axis-parallel hyperellipsoid	+	+	objfun1a
rotated hyperellipsoid	+	+	objfun1b
Rosenbrock	+	+	objfun2
Rastrigin	+	+	objfun6
Schwefel	+	+	objfun7
Griewangk	+	+	objfun8
sum of powers	+	+	objfun9
double integrator		+	objdopi
harvest problem		+	objharv
discrete LQ		+	objlinq
continuous LQ		+	objlinq2
push cart problem		+	objpush

2.10 Application considerations

In several (technical and non-technical) areas of life, frequently arise optimum problems, which must be solved by experts. Optimum problems can be static and dynamic. The goal of optimization can be the optimization (minimizing or maximizing) of a scalar criterion. In case of a maximum problem the optimization goal multiplied by (-1) can be chosen and the minimum of the new objective function can be searched.

In the problem restrictions can be specified. Since genetic algorithms expect the x_i variables to fall into a limited region, e.g. $lb_i \le x_i \le lu_i$, these restrictions don't make trouble in case of genetic algorithms. Other types of restrictions, e.g. the specification of the final state *x(T)* in case of dynamic optimization problems can be handled as the weighted sum of the objective function and the punishing function. When applying punishing functions, the weighted sum of the original objective function and the punishing function is treated as the scalar criterion of the optimization. The weighting factors must be selected, and the optimum is sensitive to the selection of the weighting factors.

In case of continuous-time dynamic optimization problems the interval $[t_s, t_e]$ (typically $[0, T]$) can be divided by a finite number of equidistant points, and the values of the control $u_k = u(kT_s)$ are searched. The independent variable (x) of the optimization in this case is the control $\mathbf{u} = \{u_k\}_{k=1}^{N}$, or this supplemented by the state

series $\mathbf{x} = \{x_k\}_{k=1}^N$. Because of the discretization the originally continuous-time (analogue) system must be substituted by some kind of its discrete-time (sampled) equivalent (in the linear case) or approximation (in the nonlinear case).

In case of dynamic optimization problems by using GA, as at every other optimization method, the large computation time of the objective function (*objfun*) causes a trouble in the case of real problems and simulation techniques, because usually large scale state equations must numerically be solved to determine the state transients.

If the model of the system is unknown, the simulation technique is not applicable. In this case the real system must be experienced in real-time, which rarely accepts such a control during the optimization process, which significantly differs from the best previous one. Since, due to the recombination and mutation, the population can contain wrong individuals (controllers) too, hence there is a serious danger of giving such a control to the real system which can cause big dynamic and transient errors or unstable work, prohibited during normal system operation. The early detection and aborting of these transients is of key importance. In such situation the system must be brought back to the normal operation, then a new experience can be performed to choose a suitable individual (controller). However, this is partly in opposition with the original concept of the genetic algorithm.

At several problems more scalar criteria must be simultaneously optimized (non-scalar valued, e.g. vector, matrix, etc. problems). This time the sharp difference must be noted between the minimum and the infimum, since not all objective function values are comparable with each other. Let $Q \subset E$ be a set in the space *E*, in which a partial ordering ≥ is defined. It is declared that $y_0 = \min Q$ is a minimal solution, if $y_0 \in Q$ and there exists no $y \in Q$ for which $y_0 \geq y$ and $y_0 \neq y$. There can be several minima, since not all y, y_0 pairs are comparable (≥ is only a partial ordering). On the contrary, the infimum is stronger, however rarely exists. It is declared that $y_0 = \inf Q$, if for all $y \in Q$ it is satisfied $y \geq y_0$. It must be noted that if the infimum exists, then every y, y_0 pair must be comparable.

If several scalar criteria must be simultaneously optimized, then it is a vector criterion problem. The partial ordering is defined using the notations $\mathbf{y}_1 = (y_{11},...,y_{1N})^T$ and $\mathbf{y}_2 = (y_{21},...,y_{2N})^T$: $\mathbf{y}_1 \geq \mathbf{y}_2$, if $y_{1i} \geq y_{2i}$ for all *i=1,...,N*.

In an *N*-person nonzero-sum game each of the *N* players wants to choose such a strategy which minimizes his/her cost function. If there is no severe hostility between the players, they can form a common interest alliance (co-operation) and choose such a common x_0 strategy, for which in the case of any other *x* strategy the cost of each player equals to the cost belonging to x_0, or at least one player is punished so that his/her cost raises compared to the cost of x_0. This kind of strategy is

called Pareto-optimum. Let's note that due to the co-operation it is enough to consider only one player, whose $\mathbf{F}_0(x)=(F_{01}(x),...,F_{0N}(x))^T$ cost function is vector-valued, where $F_{0i}(x)$ would be the i^{th} player's cost function. Thus the vector space $E=R^N$ can be considered with the partial ordering of vectors. The aforementioned condition of the Pareto-optimum exactly means that in case of $x\neq x_0$ and $\mathbf{F}_0(x)\neq\mathbf{F}_0(x_0)$, $\mathbf{F}_0(x_0)\geq\mathbf{F}_0(x)$ can not happen (at least one player is punished). Thus x_0 is Pareto-optimal strategy $\Leftrightarrow \mathbf{F}_0(x_0)=\min_x \mathbf{F}_0(x)$.

The necessary condition of the Pareto-optimum according to the theory is that $l_i\geq 0$, i=1,...,N exist so that $\sum_{i=1}^{N} l_i F_i'(x_0)=0$. The positive weights $l_i>0$ can be selected and the optimum of the scalar-valued criterion $\sum_{i=1}^{N} l_i F_{0i}(x)$ can be considered to determine a Pareto-optimum. Unfortunately we usually do not know which l-choice belongs to a preferable Pareto-optimum.

Other approaches are also possible in the case of vector-optimum problems, especially if the set of $F_{0i}(x)$ objective values can be substituted by a scalar-valued fitness function.