A Method for Optimizing Fuzzy Membership Functions

Ludovic Koehl and Xianyi Zeng

GEMTEX
Ecole Nationale Superiéure des Arts
et Industries Textiles (ENSAIT)
BP 30329
59056 Roubaix CEDEX
FRANCE
e-mail: Xian.Zeng@univ-lille 1.fr

Christian Vasseur

Laboratoire d'Automatique I3D
Bât. P2, Cité Scientifique
59650 Villeneuve d'Ascq
FRANCE

Abstract: This paper presents a new method for optimizing both the shape and the internal parameters of fuzzy membership functions. Given a specific shape of membership functions, the corresponding internal parameters are optimized by a genetic algorithm. The overall optimization of membership functions is done using a learning automaton coupled with several genetic algorithms each running in its own parameters space. At each step, this learning automaton permits to randomly select the next parameters space according to the behavior evaluation of the current genetic algorithm.

Keywords: Fuzzy Logic Controllers, Optimization, Genetic Algorithm, Learning Automaton.

Ludovic Koehl received the diplôme d'Ingénieur Textile from the ENSAIT (Ecole Nationale Supérieure des Arts et Industries Textiles), Roubaix, France, in 1994 and the Doctor degree in Automation and Industrial Computer Science from the Université de Lille I, Villeneuve d'Ascq, in 1998. He is currently a research/teaching assistant in the ENSAIT. His research interests include fuzzy modeling, genetic algorithms, fractal geometry and their applications in textile industry.

Xianyi Zeng received the BSc. degree in Computer Science from the Tsinghua University, Beijing, R.P.China, in 1986 and the Doctor degree in Automation and Industrial Computer Science from the Université de Lille I, Villeneuve d'Ascq, in 1992. He is currently an Associate Professor at the ENSAIT. His research interests include fuzzy modeling, genetic algorithms, fractal geometry and their applications in textile industry.

Christian Vasseur received the diplôme d'Ingénieur from the Ecole Centrale de Lille, Villeneuve d'Ascq, France, in 1970 and the Doctor degree in Automation and Industrial Computer Science from the Université de Lille I, Villeneuve d'Ascq, in 1972. He is currently Professor at the University de Lille I and leads the Laboratoire d'Automatique I3D. His research interests include pattern recognition, control of complex processes and signal processing.

1. Introduction

A large number of contributions has recently been devoted to modeling nonlinear systems using Fuzzy Logic Controllers (FLCs). Theoretically, FLCs have been proved to be universal approximators under very weak assumptions, i.e. they are capable of approximating any real continuous function on a compact set to arbitrary accuracy [Wan 92a] [Cas 95]. In practice, many industrial problems have been successfully solved using FLCs.

A FLC has the characteristic of representing human knowledge or experiences as fuzzy rules. However, in most of the existing FLCs, shapes and internal parameters of membership functions and fuzzy rules are determined and tuned through trial and error by operators. Therefore, it is necessary to design FLCs so that these elements can be optimized [Shi 95]. Some self-tuning methods have been proposed to solve this problem using neural networks, the backpropagation algorithm, genetic algorithms, etc. However, these methods often suppose that the number and shapes of membership functions are defined a priori. Then, the extracted fuzzy rules and fuzzy partitions in the input space and the output space are not globally optimal due to these constraints.

In this paper, we present a new method for self-tuning together the shape and internal parameters of membership functions. This fuzzy model is designed in order to approximate a multi-inputs/single output continuous function where the relationship between input variables and output variable is unknown. Our paper is organized as follows.

First, we present a procedure given in [Abe 95] for extracting fuzzy rules directly from the numerical data of the knowledge base. In this procedure, the universe of the output is divided into a number of intervals. By putting these data into different classes according to the output intervals, we define two kinds of regions in the input space: activation hyperboxes and inhibition hyperboxes. For a given class of input data, an activation hyperbox contains all data belonging to this class and an inhibition hyperbox inhibits the existence of data for this class. Inhibition hyperboxes can be located by finding overlaps between neighboring activation

hyperboxes. In these located inhibition hyperboxes can be defined new activation and inhibition hyperboxes for the next level. This procedure is repeated until overlaps are solved.

In this paper, the input variables are assumed to be non-redundant, i.e. the total number of fuzzy rules for a given number of classes decreases when an input variable is deleted. Thus, the fuzzy rules are defined by activation and inhibition hyperboxes. By selecting a proper Gaussian or trapezoidal function as membership function, we calculate the output value using Sugeno's defuzzification method.

By comparison with other methods of fuzzy rules extraction which assume that the space of input variables is partitioned into a number of fixed regions [Ton 80], [Wan 92b], this procedure permits to obtain more accurate fuzzy partition and fuzzy rules and it has a potential applicability to problems having a high-dimensional input space.

The second part of this paper deals with the optimization problem of parameters of the fuzzy model. The shapes and the parameters of membership functions can be optimized together using a learning automaton coupled with two genetic algorithms. The learning automaton is a stochastic approach permitting to alternate between different parameters spaces each corresponding to a genetic algorithm. The parameters optimization problem is solved using genetic algorithms because our searching space is complex and there exist many local minima. Unlike conventional search methods, genetic algorithms deal with multiple solutions simultaneously and provide global near-optimal solutions for various complex optimization problems.

At the end of this paper, several simulation results are given in order to show the effectiveness of the proposed method. This method is applied in approximating several sample functions, each of them dealing with multi-input variables and a strongly non-linear relationship between the input space and the output space.

2. Extraction of Fuzzy Rules From Data

The procedure of fuzzy rules extraction [Abe 95] is briefly presented below. The fuzzy model is considered as an universal approximator for any continuous function which has an one-dimensional output y and an n-dimensional input vector X. First we partition the universe of discourse y into m intervals as follows:

$C_1=[y_0, y_1]: y_0 \le y \le y_1$

$C_2=(y_1, y_2]: y_1 < y \le y_2$

$C_m=(y_{m-1}, y_m]: y_{n-1} < y \le y_n$

Let a set of input data for the output interval C_i be X_i, where i=1, ..., m. First, using X_i, an activation hyperbox of level 1, denoted as $A_{ii}(1)$, is defined as follows:

$A_{ii}(1)=\{X \mid v_{iik}(1) \le x_k \le V_{iik}(1)\}$

where x_k: the k-th element of input vector X;

$v_{iik}(1)$: the minimum value of x_k of $X \in X_i$; and

$V_{iik}(1)$: the maximum value of x_k of $X \in X_i$.

If there is no overlap between activation hyperboxes $A_{ii}(1)$ and $A_{jj}(1)$ ($j \ne i$, j=1, ..., m), we obtain a fuzzy rule of level 1 for the output interval i as follows:

$r_{ij}(1)$: If X is $A_{ii}(1)$ then y is C_i.

If there are some overlaps, we resolve them recursively. If an overlap exists between the activation hyperboxes $A_{ii}(1)$ and $A_{jj}(1)$, we define the overlapped region as the inhibition hyperbox of level 1 denoted by $I_{ij}(1)$:

$I_{ij}(1) = A_{ii}(1) \cap A_{jj}(1)$

Denoting $w_{ijk}(1)$ and $W_{ijk}(1)$ the lower and the upper bounds of I_{ij} on the axis x_k, we have $v_{iik}(1) \le w_{ijk}(1) \le W_{ijk}(1) \le V_{iik}(1)$.

Then we define a fuzzy rule of level 1 with inhibition by

$r_{ij}(1)$: If X is $A_{ii}(1)$ and X is not $I_{ij}(1)$, then y is C_i

If some data belonging to X_i exist in $I_{ij}(1)$, the activation hyperbox of level 2 $A_{ij}(2)$ is defined within $I_{ij}(1)$. This procedure is repeated until all overlaps are resolved. At level l (≥ 2), the fuzzy rule corresponding to $A_{ij}(l)$ without inhibition is

$r_{ij}(l)$: If X is $A_{ij}(l)$, then y is C_i.

The membership degree of the fuzzy rule $r_{ij}(l)$ can be calculated by

$$d_{r_{ij}(l)}(X) = \mu_{A_{ij}(l)}(X)$$

The fuzzy rule corresponding to $A_{ij}(l)$ with inhibition is

$r_{ij}(l)$: If X is $A_{ij}(l)$ and X is not $I_{ij}(l)$, then y is C_i.

The corresponding fuzzy rule $r_{ij}(l)$ can be calculated by

$$d_{r_{ij}(l)}(X) = \max(0, \mu_{A_{ij}(l)}(X) - \mu_{I_{ij}(l)}(X))$$

where $\mu_{A_{ij}(l)}(X)$ and $\mu_{I_{ij}(l)}(X)$ are the membership degrees of the activation hyperbox $A_{ij}(l)$ and the inhibition hyperbox $I_{ij}(l)$ defined below.

The procedure of fuzzy rules extraction is illustrated schematically in Figure 1.

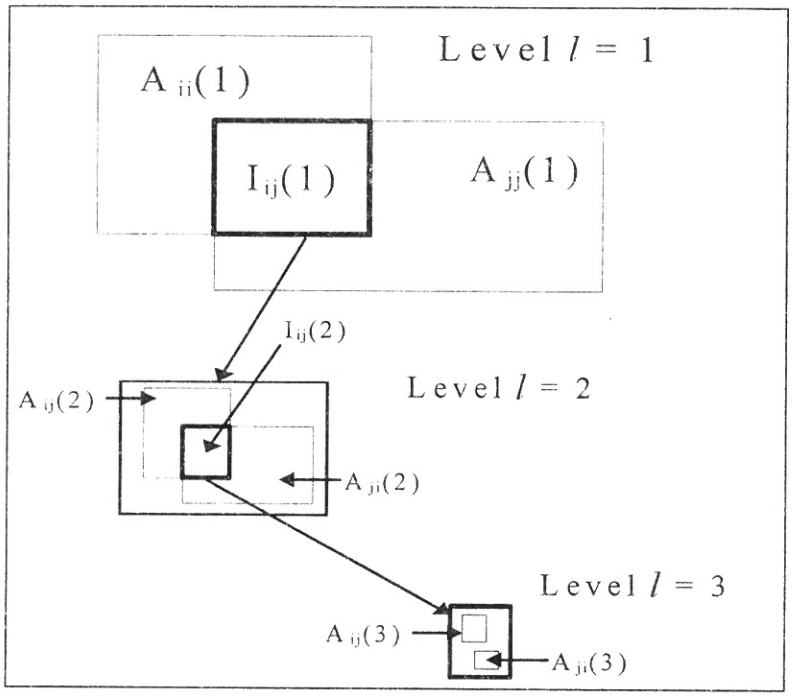

Figure 1. Recursive Definition of Activation and Inhibition Hyperboxes

The membership degree of each input vector X with respect to a hyperbox Z ($A_{ij}(l)$ or $I_{ij}(l)$) is defined by

$$\mu_Z(X) = \min_{k=1,\ldots,n} \left\{ \mu_Z(x_k, \gamma_k) \right\}$$

$\mu_Z(x_k, \gamma_k)$ is the membership degree for each element of X: x_k. It is defined as a function of three parameters: u_k, U_k and γ_k. The interval $[u_k, U_k]$ is the projection of the hyperbox Z on the axis x_k and γ_k is a sensitivity parameter.

In this paper, the membership degree for each input variable adopts one of the two shapes: 1) symmetric trapezoidal shape; 2) Gaussian shape. The trapezoidal membership function (see Figure 2) is defined as follows.

$$\mu_Z(x_k, \gamma_k) = \begin{cases} 1 & x_k \in [u_k, U_k] \\ 1 - \max(0, \min(1, \gamma_k(u_k - x_k))) & x_k \prec u_k \\ 1 - \max(0, \min(1, \gamma_k(x_k - U_k))) & x_k \succ U_k \end{cases}$$

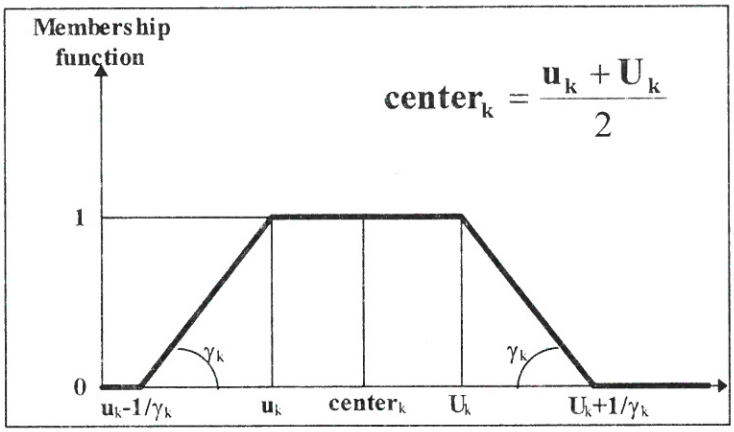

Figure 2. Trapezoidal Membership Function for $\mu_Z(x_k, \gamma_k)$

The Gaussian membership function (Figure 3) is defined as follows.

$$\mu_Z(x_k, \gamma_k) = \exp\left(-\frac{1}{2}\cdot\frac{\left(x_k - \text{center}_k\right)^2}{\gamma_k^2}\right)$$

$$\text{center}_k = \frac{u_k + U_k}{2}$$

with $\gamma_k \neq 0$ for any $k \in \{1, 2, ..., n\}$

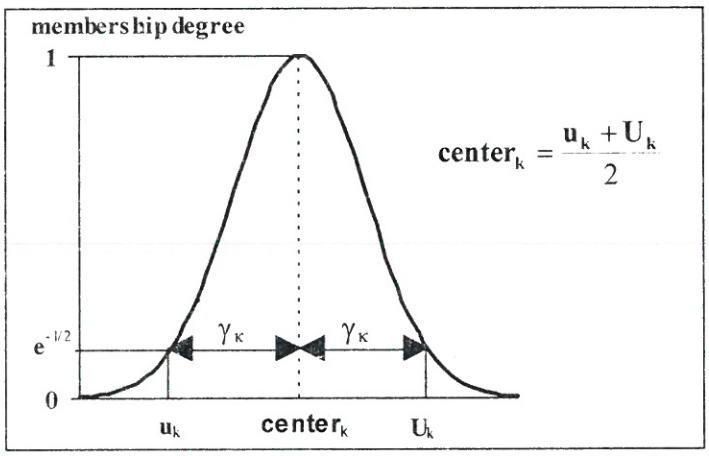

Figure 3. Gaussian Membership Function for $\mu_Z(x_k, \gamma_k)$

The final membership degree of X with respect to a set of fuzzy rules $\{r_{ij}(l) \mid l=1, ...\}$ denoted as $d_{r_{ij}(l)}(X)$ is given by

$$d_{r_{ij}}(X) = \max_{l=1,...}(\mu_{r_{ij}(l)}(X))$$

The membership degree of X with respect to the output interval C_i denoted by $d_i(X)$ is given by

$$d_i(X) = \min_{\substack{j\neq i, j=1,...,n \\ A_{ii}(1)\cap A_{jj}(1)\neq\Phi}} (d_{r_{ij}}(X))$$

By assuming that y obeys a pre-defined membership function in each output interval C_i, the defuzzification procedure permits to calculate the approximative output $\hat{y}$ as follows.

$$\hat{y} = \frac{\sum_{i=1}^{m} d_i(X) \cdot m_i \cdot \sigma_i}{\sum_{i=1}^{m} \sigma_i \cdot d_i(X)}$$

where m_i and σ_i are the mean and the variance of the membership degree corresponding to C_i.

The main elements affecting the precision of $\hat{y}$ include the shapes of membership functions of $\mu_Z(x_k, \gamma_k)$ for all x_k and the corresponding parameters u_k, U_k and γ_k. These parameters constitute a very complex high dimensional function of the approximative output $\hat{y}$. The optimization of these parameters as well as the decomposition of this complex fuzzy model have been done using genetic algorithms.

3. Genetic Algorithms

Genetic algorithms have shown increasing interest in engineering applications since the publication of Holland's paper in 1975 [Hol 75]. Genetic algorithms are designed to find a global maximum of a function of many variables by performing a particular kind of genetic-inclined search in the space of these variables. This form of search is especially interesting when solving complex optimization tasks, including the structure and parameters optimization problem [Ped 96]. The main principle of a genetic algorithm is described as follows.

It begins with an initial set (also called population) of randomly generated potential solutions to an optimization problem. The value of a fitness function is evaluated for each solution, and the "best" solutions are selected for survival according to the selection probabilities. For each solution, the selection probability is defined as a function of its fitness value. Then, the genetic algorithm manipulates these selected solutions in its search for better solutions. Each solution is encoded into a binary string, so that new encoded solutions can be generated through the exchange of information among surviving solutions (crossovers) as well as sporadic alternations in the bit string encodings of the solutions (mutations).

According to [Mic 94], the power of genetic algorithms is in their capacity of simultaneously exploring several regions in the space of potential solutions. They can be run without a learning base. During the running of a genetic algorithm, the search concentrates progressively on the regions where the optimum is located in.

Typically, a genetic algorithm is characterized by the following components [Mic 94]:

1) a string representation for the feasible solutions to the optimization problem,

2) a population of encoded solutions,

3) a fitness function that evaluates each solution,

4) genetic operators that generate a new population from the existing population.

In this paper, genetic algorithms are applied for solving the parameters optimization problem in a fuzzy model where fuzzy rules and fuzzy partitions have been obtained according to the method presented in Section 2. Our objective is to find the best shapes and the most appropriate parameters of the membership functions so that the total error between the outputs of fuzzy models and desired outputs derived from the learning data is minimized.

4. Optimization of Membership Functions

In our paper the membership functions are limited to symmetric trapezoidal shape and

Gaussian shape. A space of internal parameters (U_k, u_k, γ_k) defined in Section 2 is associated with each shape. In order to obtain the overall optimum of membership functions, we have to consider both parameters optimization and shape optimization. One efficient way to do so is the application of two genetic algorithms together each running in one parameters space for its optimization. In this case, it is necessary to define an adaptive mechanism to alternate between these two parameters spaces according to the environment or the reward/penalty signal emitted from the running of the current genetic algorithm.

In this paper we use a learning automaton coupled with two genetic algorithms to realize this function. A learning automaton is a stochastic automaton in feedback connection with a random environment [Nar 74], [Nar 89]. The output of the automaton (actions) is the input to the environment and the output of the environment (responses) is the input to the automation

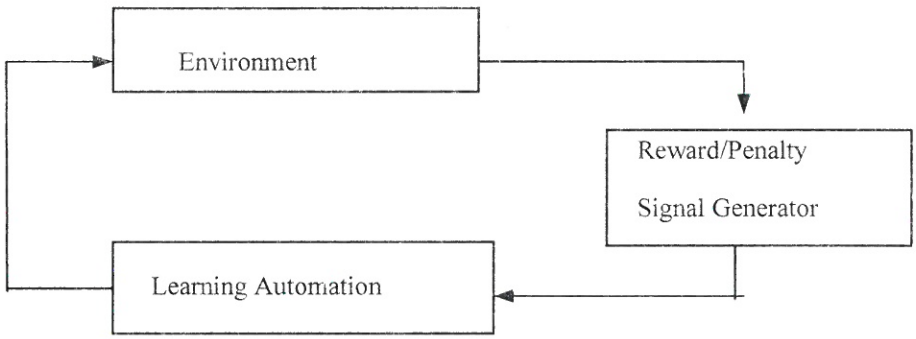

Figure 4. Learning Automaton and Its Environment

In general, a learning automaton (Figure 4) is defined by (A, Q, R, T) and the environment by (A, R, D), where

$A=\{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the set of all actions of the automaton. The action of the automaton at period k is denoted by $\alpha(k)$ and $\alpha(k)\in A$ for k=0, 1, 2, ... Evidently, A is the set of outputs of the automaton and it is also the set of inputs to the environment.

R is the domain of responses from the environment. Let $\beta(k)$ denote the response/penalty signal received by the automaton at period k where $\beta(k)\in R$, $\forall k$. $\beta(k)$ is the output of the environment at period k and it is also the input to the automaton.

$D=\{d_1, d_2, ..., d_r\}$ is the set of reward probabilities, where

$d_i(k)=E[\beta(k) \mid \alpha(k)=\alpha_i]$

If the d_i's are independent of *k*, the environment is called stationary. Otherwise, it is nonstationary. The reward probabilities are unknown to the automaton.

Q is the state of the automaton defined by

$Q(k)=[\ P(k),\ \hat{D}(k)\ ]$

where $P(K)=[p_1(k), ..., p_r(k)]$

($\forall k$, we have $0\le p_i\le 1$, $\sum_{i=1}^{r} p_i(k)=1$)

is the action probability vector and

$$\hat{D}(k)=\left[\hat{d}_1(k),.....,\hat{d}_r(k)\right]$$

is the vector of estimates of the reward probabilities at the k-th period.

T is the learning algorithm or the reinforcement scheme which is used by the automaton in order to update its state. At period k we get from T

$Q(k+1)=T(Q(k), \alpha(k), \beta(k))$

During the running of algorithm T, the automaton randomly selects an action $\alpha(k)$ from the set of actions A at each period k. The selection of actions depends on their current action probability vector P(k). The selected action $\alpha(k)$ becomes input to the environment and the environment gives to the input of the automaton a random response $\beta(k)$ whose expected value is d_i if $\alpha(k)=\alpha_i$. Next, the

automaton calculates Q(k+1) using the reinforcement scheme T.

The previous procedure is repeated until the optimal action on the environment is found. In this case, we have

$d_m = \max_j \{d_j\}$ where α_m is the optimal action.

The action α_m has the maximum probability of being rewarded. It is desired that the action probability corresponding to α_m (i.e. p_m) tends to unity as the time k goes to infinity.

For simplicity of notations, we do not distinguish between the reward probabilities and their estimates in the following discussion.

In our optimization procedure, two shapes of membership function are available. So, the learning automaton is designed to have the following two actions (r=2): α_1=symmetric trapezoidal membership function, α_2=Gaussian membership function. Each action is associated with one genetic algorithm for optimizing its corresponding parameters of membership functions (u_k, U_k, γ_k). At each period k, the genetic algorithm runs independently in its parameters space for five generations.

The behavior of each genetic algorithm can be evaluated from the total error of the learning input/output data $E=\frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$ (N is the total number of learning data). The error E is a function of the parameters (u_k, U_k, γ_k) (k=1, ..., m) corresponding to the current genetic algorithm because the value of each $\hat{y}_i$ is calculated from the fuzzy model associated with the specific fuzzy parameters. In each genetic algorithm, E is taken as fitness function and the encoding scheme is given as follows.

1) For trapezoidal membership functions:

For a given input variable x_k ($k \in \{1, ..., n\}$) and a given output interval C_i, the parameters U_k, u_k are rewritten by

$U_k=V_{iik}(l)+s_k(V_{iik}(l)-v_{iik}(l))$ and $u_k=v_{iik}(l)+q_k(V_{iik}(l)-v_{iik}(l))$ for the activation hyperbox A_{ii} when l=i and

$U_k=V_{ijk}(l)+s_k(V_{ijk}(l)-v_{ijk}(l))$ and $u_k=v_{ijk}(l)+q_k(V_{ijk}(l)-v_{ijk}(l))$ for the activation hyperbox A_{ij} when l>1.

$U_k=W_{ijk}(l)+s_k(W_{ijk}(l)-w_{ijk}(l))$ and $u_k=w_{ijk}(l)+q_k(W_{ijk}(l)-w_{ijk}(l))$ for the inhibition hyperbox I_{ij}.

where s_k, q_k are real numbers varying between -0.1 and 0.1, which adjust the upper bound and the lower bound on the axis x_k of the corresponding hyperbox.

For simplicity, we adopt the same values of s_k's and q_k's for different levels of hyperboxes and different output intervals.

The parameter γ_k is rewritten by

$\gamma_k=\tan(\alpha_k)$ where α_k is associated with the input variable x_k and varies between $\pi/4$ and $\pi/2$, which adjusts the slope of the corresponding membership function.

Equally partitioning the universe of discourse of each α_k into 16 subintervals and that of each parameter of the s_k's and the q_k's into 4 subintervals, we encode these parameters into 4 bits and 2 bits respectively. Then, we obtain 8n bits for each solution of the searching space.

2) For Gaussian membership functions:

U_k and u_k are calculated from $U_k=center_k+\gamma_k$ and $u_k=center_k-\gamma_k$

$center_k$ is the center of the corresponding hyperbox on the axis x_k with

$center_k=(V_{ijk}(l)+v_{ijk}(l))/2+s_{ki}(V_{ijk}(l)-v_{ijk}(l))$ for the activation hyperbox A_{ij} at level l (i=j when l=1 and i≠j when l>1) or $center_k=(W_{ijk}(l)+w_{iik}(l))/2+s_{ki}(W_{iik}(l)-w_{iik}(l))$ for the inhibition hyperbox I_{ij} at level l.

s_k is a real number varying between -0.1 and 0.1, which adjusts the center's position of the corresponding membership function.

γ_k varies between 0 and its maximal value γ, which adjusts the variance of the membership function.

Equally partitioning the universe of discourse of each s_k into 4 subintervals and that of each γ_k into 16 subintervals, we encode them into 2 bits and 4 bits respectively. Then, we obtain 6n bits for each solution of the searching space.

The optimization of membership functions leads us to search for the overall optimal solution for both of the parameters spaces. The learning automaton is used to alternate dynamically

between these searching spaces in order to select the best shape of membership functions.

At each period of the learning automaton, the response of the environment to the action $\alpha(k)$ is an evaluation function F(k), defined as the minimal value of E for all chromosomes in the five generations.

The reward/penalty signal $\beta(k)$ is designed to vary in the interval [0, 1] and to take into account two elements characterizing the environment: 1) the stability S(k), defined by $S(k)=|F(k)-F(k-1)|$; 2) optimum F(k). In a genetic algorithm, if the fitness value of the best chromosome varies little through several generations, i.e. the value of S(k) is small, then the action $\alpha(k)$ is considered as stable and close to the optimal action a_m. Moreover, the value of $\beta(k)$ is wished to be decreased by a bad action ($F(k)>F(k-1)$) (penalty) and to be increased by a good action ($F(k)<F(k-1)$) (reward). According to these principles, $\beta(k)$ is defined by

$$\beta(k)=\frac{2}{\pi}\arctan\left(C^{-1}(k)\right)$$

where $C(k)=S(k)+\lambda F(k)$ and λ is a positive coefficient defined by operators.

The criterion C is a linear combination of the stability S(k) and the optimum F(k). The evolution of chromosomes through a number of generations permits to obtain stable values of S(k) and F(k).

In the learning automaton, we take the Pursuit Algorithm [Oom 90] as a reinforcement scheme. It is illustrated as follows:

Step 0: *k:=0*;

Step 1 (initiation): set $X_i(k)=0$, $n_i(k)=0$ for i=1, ..., r.

Step 2: Set $P_i(k)=1/r$ for $1\le i\le r$. Initialize d(k) by picking up each action for a small number of times and setting $d_i(k)$ to the average reaction obtained.

Step 3: At period k ($k\ge 0$), $\alpha(k)=\alpha_i$ is selected according to the distribution of the action probabilities P(k).

Step 4: Apply the genetic algorithm corresponding to $\alpha(k)$ It runs based on the last population obtained by the same action. By taking into account the response $\beta(k)$, we adopt the following reinforcement scheme.

Update action probabilities P(k): $P(k+1)=P(k)+\mu(e_{M(k)}-P(k))$

where e_j is an r-dimensional vector with j-th component unity and all others zero, M(k) is the index of the maximal reward estimate, i.e.

$d_{M(k)}=\max\{d_j(k) \mid j=1, 2, ..., r\}$

and μ is the internal parameter of the learning algorithm.

Update reward probabilities D(k):

$X_i(k+1)=X_i(k)+\beta(k)$

$n_i(k+1)=n_i(k)+1$

$X_j(k+1)=X_j(k)$ for $j\ne i$

$n_j(k+1)=n_j(k)$ for $j\ne i$

$\forall j\in\{1, ..., r\}$, we have $d_j(k+1)=X_j(k+1)/n_j(k+1)$

The previous updated automaton elements are used at period k+1.

The Pursuit Algorithm used permits to quickly converge to the overall optimum by selecting the best action. Finally, we obtain the most appropriate shape and the most pertinent parameters of membership functions. The internal parameters of the fuzzy model in both spaces are optimized alternatively by the learning automaton coupled with two genetic algorithms. The general scheme of the optimization of membership functions is given in Figure 5.

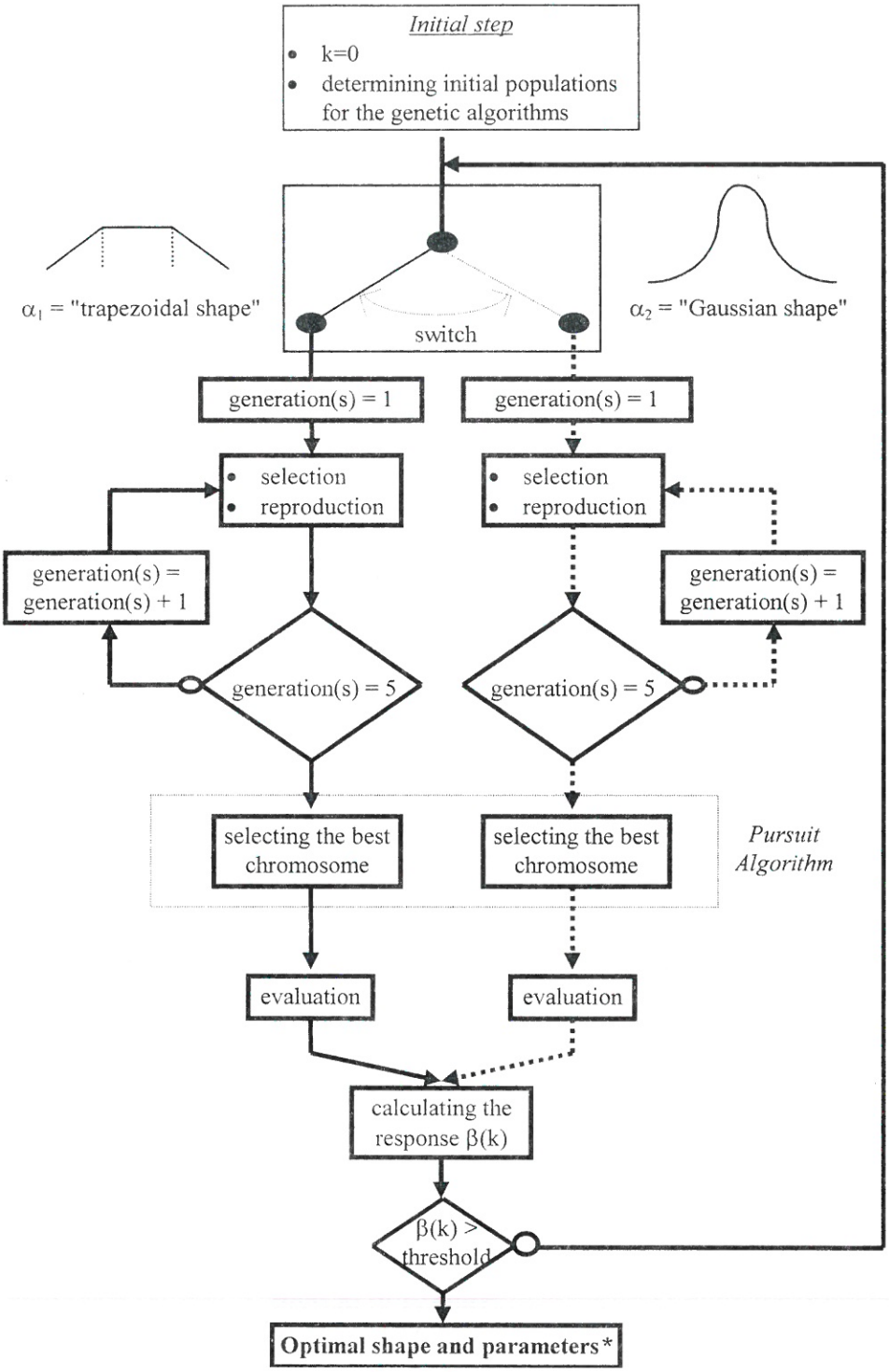

Figure 5. General Scheme of the Optimization of Membership Functions

5. Simulation Results

In order to show the effectiveness of the proposed method, we apply it for approximating two multi-input nonlinear functions.

Example 1: approximation of the function

$$y=\frac{1}{12}(4x_1x_2+x_3x_4+x_5) \text{ with } x_1, \ldots, x_5 \in [0, 1]$$

In this example, we take uniformly 5 samples for each input variable. Then, we obtain $5^5=3125$ input/output data from this function for

building the learning base Ω, i.e. $\Omega=\{(X_i, y_i) \mid i=1, ..., 3125\}$ where X_i is the i^{th} input vector and y_i the output value corresponding to X_i. The interval of the output variable y [0, 0.5] is equally partitioned into 8 classes as follows.

C_1=[0, 0.0625], C_2=]0.0625, 0.125], C_3=]0.125, 0.1875], C_4=]0.1875, 0.25],

C_5=]0.25, 0.3125], C_6=]0.3125, 0.375], C_7=]0.375, 0.4375], C_8=]0.4375, 0.5].

The uniform partition of y leads to defining for each output interval C_i the mean value m_i and the variance σ_i in an uniform way:

$$m_i=\frac{1}{2}\left(C_i^{min}+C_i^{max}\right),$$

$$\sigma_i=\frac{1}{2}\left(C_i^{max}-C_i^{min}\right)$$

where C_i^{min} and C_i^{max} are lower and upper bounds of C_i respectively.

The criterion of optimization E is calculated by

$$E=\frac{1}{3125}\sum_{i=1}^{3125}\left|y_i-\hat{y}_i\right|$$

In genetic algorithms, the parameters are selected as follows:

- the crossover probability p_c=0.25;
- the mutation probability p_m=0.01;
- population size = 20

Running genetic algorithms leads to the following results. Fig-6 shows the evolution of reward probabilities varying with time. After the learning automaton has run for 120 periods, the probability corresponding to the trapezoidal shape is higher than 0.9 and tends towards 1. So, it is finally taken as the optimal shape of a membership function.

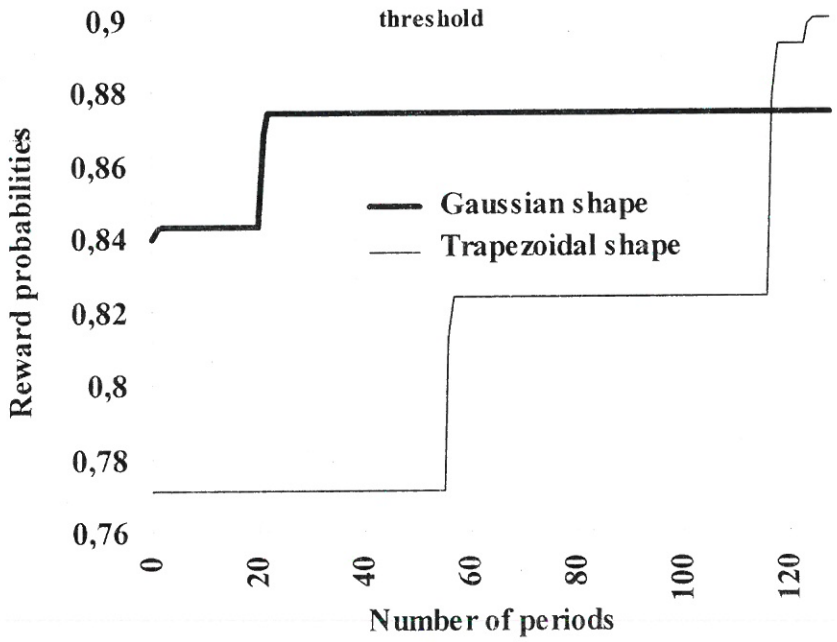

Figure 6. Evolution of Reward Probabilities for Example 1

Having run the learning automaton for optimizing the shape and the internal parameters of membership functions, we calculate the averaged error E between the optimized fuzzy model and the original function. We obtain: E=0.0032117.

Example 2: approximation of the function

$$y=\frac{1}{14}\left(4x_1x_2+x_3x_4+2\sin\left(\frac{\pi}{2}x_5\right)\right)$$

with $x_1, ..., x_5 \in [0, 1]$.

In this example, we also take 3125 input/output learning data from the function. The simulation results are shown as follows.

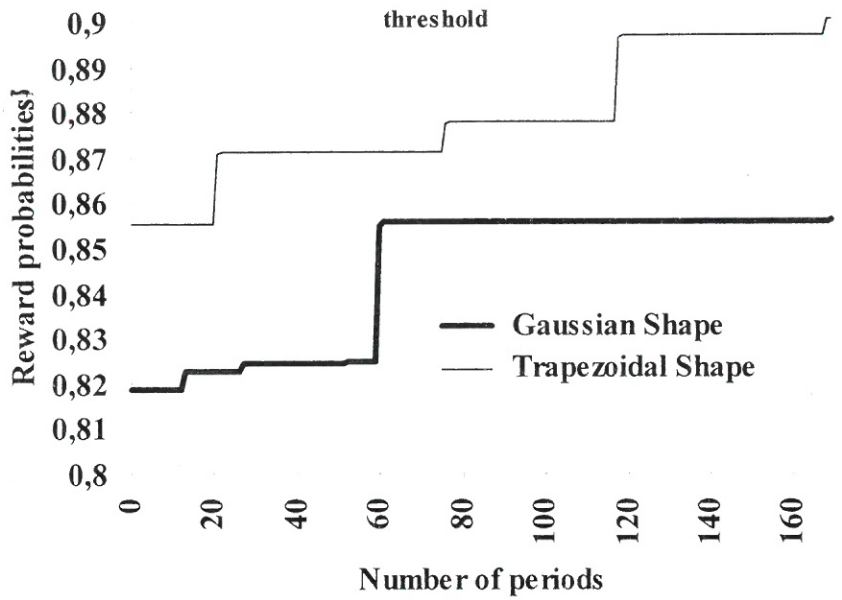

Figure 7. Evolution of Reward Probabilities for Example 2

As in Example 1, the trapezoidal shape is the best shape of membership functions. After the optimization, we obtain the averaged error as follows: E=0,003862. We can see from these two examples that the optimized fuzzy model provides us with the results very close to those of the original functions. So, the effectiveness of the proposed method is verified.

6. Conclusion

As many existing works did, the optimization of membership functions of our fuzzy model was solved by genetic algorithms, which are considered as very efficient methods dealing with complex optimization problems. According to our experience, the behavior of genetic algorithms is strongly related to the size of the searching space. It is not necessary to apply genetic algorithms if the number of solutions is too small and meanwhile it is difficult to obtain solutions close to the optimum if the searching space is too large or the continuity of the searching space is not taken into account by the encoding scheme of chromosomes. This constraint leads to decomposing big size complex optimization problems into several smaller subproblems and to designing a mechanism such as learning automata which alternates between the corresponding subspaces in order to find the global optimum.

REFERENCES

[Abe 95] ABE, S. and LAN, M., **Fuzzy Rules Extraction Directly From Numerical Data for Function Approximation**, IEEE TRANS. ON SMC, Vol.25, No.1, January 1995, pp.119-129.

[Cas 95] CASTRO, J. L., **Fuzzy Logic Controllers Are Universal Approximators**, IEEE TRANS. ON SMC, Vol.25, No.4, 1995, pp.629-634.

[Hol 75] HOLLAND, J.H., **Adaptation in Natural and Artificial Systems**, MIT PRESS, 1975.

[Mic94] MICHALEWICZ, Z., **Genetic Algorithms + Data Structure = Evolution Programs**, SPRINGER- VERLAG, Berlin, 1994.

[Nar74] NARENDRA, K. S. and THATHCHAR, M.A.L., **Learning Automata - A Survey**, IEEE TRANS. ON SMC, Vol. 14, 1974, pp. 323-334.

[Nar89] NARENDRA, K.S. and THATHCHAR M.A.L., **Learning Automata: An Introduction**, PRENTICE HALL, Englewood Cliffs, NJ, 1989.

[Oom90] OOMMEN, B.J. and LANCTOT, J.K., **Discretized Pursuit Algorithm Learning Automata**, IEEE TRANS. ON SMC, Vol.20, 1990, pp.931-938.

[Ped96] PEDRYCZ, W. and OLIVEIRA, J.V., **An Algorithm Framework for Development and Optimization of Fuzzy Models**, FUZZY SETS AND SYSTEMS, Vol. 80, 1996, pp.37-55.

[Shi95] SHIMOJIMA, K., FUKUDA,T. and HASEGAWA, Y., **Self-tuning Fuzzy Modeling with Adaptive Membership Functions, Rules, and Hierarchical Structure Based On Genetic Algorithm**, FUZZY SETS AND SYSTEMS, Vol.71, 1995, pp.295-309.

[Ton80] TONG, R.M., **The Evaluation of Fuzzy Models Derived From Experimental Data**, FUZZY SETS AND SYSTEMS, Vol.4, 1980, pp.1-12.

[Wan92a] WANG, L. X., **Fuzzy Systems Are Universal Approximators**, Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, CA, 1992, pp. 1163-1170.

[Wan92b] WANG, L. X and MENDEL, J.M., **Generating Fuzzy Rules By Learning From Examples**, IEEE TRANS. ON SMC, Vol.22, No.6, 1992, pp.1414-1427.