First publ. in: Intelligent Data Analysis 3 (1999), 1, pp. 37-53

Constructing fuzzy graphs from examples

Michael R. Berthold [a,*], Klaus-Peter Huber [b,1]

[a] *Berkeley Initiative in Soft Computing, Computer Science Division, Department of EECS, 329 Soda Hall, University of California, Berkeley, CA 94720, USA*

[b] *University of Karlsruhe, Am Zirkel 2, 76128 Karlsruhe, Germany*

Abstract

Methods to build function approximators from example data have gained considerable interest in the past. Especially methodologies that build models that allow an interpretation have attracted attention. Most existing algorithms, however, are either complicated to use or infeasible for high-dimensional problems. This article presents an efficient and easy to use algorithm to construct fuzzy graphs from example data. The resulting fuzzy graphs are based on locally independent fuzzy rules that operate solely on selected, important attributes. This enables the application of these fuzzy graphs also to problems in high dimensional spaces. Using illustrative examples and a real world data set it is demonstrated how the resulting fuzzy graphs offer quick insights into the structure of the example data, that is, the underlying model. The underlying algorithm is demonstrated using several Java applets, which can be found under 'Electronic annexes' on www.elsevier.com/locate/ida.

Keywords: Fuzzy Graphs; Learning; Rule Extraction; Function Approximation; Interpretation

1. Introduction

Building function approximators from training examples that allow an additional ‘‘look inside’’ has gained importance with the growing interest in real world applications. One important area of interest is the generation of models from observations of complex systems, where no experts are available to describe the underlying behaviour. Problems that arise in these and other scenarios are large amounts of data and noisy measurements, as well as the need for an interpretation of the resulting model. Several methods to attack this problem have been proposed in the field of intelligent data analysis, [11] shows an interesting snapshot.

Neural network based methods have proven to show good performance but most methods lack a way to interpret the resulting model. Methods to extract rules from neural networks are usually time-consuming and hard to apply (see for example Ref. [12,21,22] or for a method that extracts fuzzy rules [23]). Extracting rules directly from the data is usually complicated, especially in the

* Corresponding author. Tel.: +1 510 642 9827; fax: +1 510 643 7684; e-mail: berthold@cs.berkeley.edu; website: WWW:http://cs.berkeley.edu/~berthold.

1 E-mail: KlausPeter.Huber@Informatik.Uni-Karlsruhe.DE

case of noisy data examples, due to the crisp nature of the underlying rules [17,25]. Decision trees offer an alternative, but are usually harder to understand [5,15,16].

Directly learning fuzzy rule bases offers an interesting alternative. Algorithms that adjust an a-priori defined rule set have been proposed before, but the structure of the rule base has to be defined by an expert (one example can be found in Ref. [20]). Automatically finding fuzzy rules is therefore of interest, but most known systems produce a large rule base, because they construct a global grid that partitions the entire input space. The method proposed in Ref. [24] for example requires an a-priori known number of membership functions for the input variables. The feature space is divided into equally spaced tiles, according to the user defined range and number of partitions for each attribute and the output variable. This results in a predefined grid of rules and the training algorithm simply assigns the output value of the closest pattern in the training data to each rule. This leads to problems as soon as the approximated function shows extrema that do not fit onto that grid. In contrast the method proposed in Ref. [8] intends to solve this problem by inserting new membership functions on the input or output variables at the point of maximum error. This process has to be stopped either using a maximum number of membership functions or a minimum error boundary. Focusing on points of maximum error results, however, in a preference to model outliers.

Several other systems were proposed that fine tune a predefined set of rules (for example see Ref. [13]). Also tree based approaches for fuzzy rules have been proposed [6]. In most cases, however, it is desired to place rules only in regions of interest. In Ref. [7] several of these methods were compared against each other, where the main result was the inability of these approaches to deal with high-dimensional feature spaces.

Zadeh [26] points out that using a non grid oriented set of rules leads to a relatively small number of fuzzy rules to characterise a complex relationship between two or more variables. An approach that tries to find individual fuzzy hyper-boxes via a growth process was proposed in Refs. [18,19] but the generation of the rule set depends heavily on the order of training examples which has to be controlled carefully to achieve satisfactory performance. An additional advantage of using models based on fuzzy logic is there ability to handle missing values, see Ref. [3] for details.

In this paper a constructive approach is presented that addresses the problem of deriving a fuzzy rule base directly from observations. The resulting rule set consists of locally independent, rectangular areas in the input space supporting trapezoidal membership functions. The combined set of soft rules or fuzzy points allows a representation in the form of a fuzzy graph. The algorithm [2] is derived from a constructive neural network training method [1] and builds this rule set from scratch requiring only a few presentations of the training patterns. In addition to crisp targets also soft numbers can be used to specify the desired output behaviour.

2. Fuzzy graphs

Zadeh illustrates fuzzy graphs in Ref. [27] as follows: “The primary function of a fuzzy graph is to serve as a representation of an imprecisely defined dependency”. Through the concept of fuzzy graphs approximate representations of functions, contours, and sets can be derived (Fig. 1, see Ref. [27] for more details).

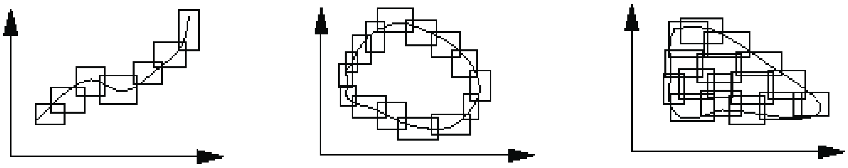

Fig. 1. Approximate representations of functions, contours, and relations (after Zadeh [27]).

In the following we will concentrate on fuzzy graphs for the approximate representation of functions, to approximate one output variable y depending on n input variables $x_i (1 \leqslant l \leqslant n)$. The typical fuzzy graphs build upon *fuzzy points* of the type

If x_1 is A_1 and $\ldots x_n$ is A_n then y is B,

where some of the input membership functions A_i can be constant, that is, $\mu_{Ai} \equiv 1$. Thus different fuzzy points may only depend on an individual set of input variables. Simplifying this equation, using $\boldsymbol{A} = A_1 \times \ldots \times A_n$ ('$\times$' denotes the Cartesian product), leads to

If x is A then y is B

which can also be expressed as a fuzzy constraint on a joint variable $(\boldsymbol{x},y)$, that is,

$(\boldsymbol{x}, y)$ is $\boldsymbol{A} \times B$.

The membership function of $\boldsymbol{A} \times B$ is given using $\wedge$ as a conjunction operator which is usually defined as the minimum

$$\mu_{\boldsymbol{A}\times B}(\boldsymbol{x}, y) = \mu_{\boldsymbol{A}}(\boldsymbol{x}) \wedge \mu_B(y) = \min\{\mu_{\boldsymbol{A}}(\boldsymbol{x}), \mu_B(y)\}$$

or, more precisely

$$\mu_{\boldsymbol{A}\times B}(\boldsymbol{x}, y) = \mu_{A1}(x_1) \wedge \ldots \wedge \mu_{An}(x_n) \wedge \mu_B(y) = \min\{\mu_{A1}(x_1), \ldots, \mu_{An}(x_n), \mu_B(y)\}.$$

Zadeh [26] uses the term *fuzzy point* to denote $\boldsymbol{A} \times B$. A collection of such rules can now be regarded as forming a superposition of m fuzzy points

$(\boldsymbol{x}, y)$ is $A_1 \times B_1 + \cdots + A_m \times B_m$,

where '+' denotes the disjunction operator (usually defined as maximum). Such a collection of rules forms a *fuzzy graph*, offering a coarse representation of a functional dependency f^* of y on $\boldsymbol{x}$. This fuzzy graph f^* can thus be defined as

$$f^* = \sum_{j=1}^{m} \boldsymbol{A}_j \times B_j.$$

Note that individual rules do not depend on a common set of membership functions on the input variables. Instead each fuzzy point is described through an individual set of membership functions $\boldsymbol{A}_j$. This makes fuzzy graphs a very compact description, especially compared to the approaches that use a "global grid", defined through only one set of membership functions for each input variable (see for example Refs. [8,24]).

The task of interpolation, that is, deriving a linguistic value B for y given an arbitrary linguistic value $\boldsymbol{A}$ for $\boldsymbol{x}$ and a fuzzy graph f^*:

$$\begin{array}{l} \boldsymbol{x} \text{ is } \boldsymbol{A} \\ \underline{f^* \text{ is } \sum \boldsymbol{A}_j \times B_j} \\ y \text{ is } B \end{array}$$

results in an intersection of the fuzzy graph f^* with a cylindrical extension of the input fuzzy set $\boldsymbol{A}$. Fig. 2 shows an example.

This functional dependency can be computed through

$$\mu_{B=f^*(A)}(y) = \sup_x \{\mu_{f^*}(\boldsymbol{x}, y) \wedge \mu_A(\boldsymbol{x})\} = \sup_x \{\mu_{A1\times B1}(\boldsymbol{x}, y) \wedge \ldots \wedge \mu_{An\times Bm}(\boldsymbol{x}, y) \wedge \mu_A(\boldsymbol{x})\}.$$

In the following a new algorithm that automatically constructs this kind of fuzzy graph from example data is explained.

3. Automatic construction of fuzzy graphs

The algorithm presented here constructs a fuzzy graph based on example data points $(\boldsymbol{x}, y)$. In addition the user specifies the granularization of the dependent variable; that is, the number and shape of the membership functions for the output variable have to be determined manually. In most applications this is no disadvantage, because it enables the user to define foci of attention or areas of interest where a finer granularization is desired. Thus c fuzzy sets are defined through membership functions μ_k^{o} ('o' for "output") for each output region k, with $1 \leqslant k \leqslant c$.

3.1. Trapezoidal fuzzy graphs

The resulting fuzzy graph consists of a set of fuzzy points or fuzzy rules R_j^k, $1 \leqslant j \leqslant m_k$, where m_k indicates the number of rules for region k. If the input space is of dimensionality n, each input

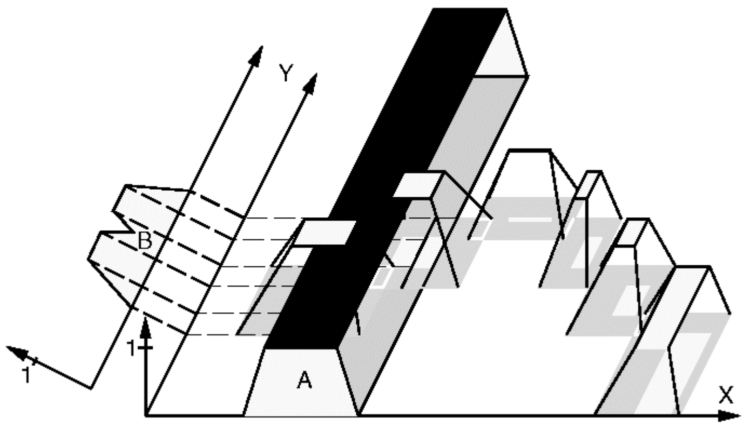

Fig. 2. Interpolation as the intersection of a fuzzy graph with a cylindrical extension of A.

vector $\boldsymbol{x}$ consists of n attributes: $\boldsymbol{x} = (x_1, \ldots, x_n)$, $x_i \in \boldsymbol{R}$ ($\boldsymbol{R}$ denotes the domain of the attributes, usually the real numbers). A rule's activity $\mu_j^k(\boldsymbol{x})$ indicates the degree of membership of the pattern $\boldsymbol{x}$ to the corresponding rule $R^k{}_{\mathrm{j}}$. Using the normal notation of fuzzy rules, each rule can be decomposed into n individual, one-dimensional membership functions:

$$\mu_j^k(\boldsymbol{x}) = \min_{i=1,\ldots,n} \{\mu_{j,i}^k(\boldsymbol{x}_i)\},$$

where $\mu_{j,i}^k$ indicates the projection of μ_j^k onto the ith attribute. The degree of membership for region k is then computed through

$$\mu^k(\boldsymbol{x}) = \max_{j=1,\ldots,m} \{\mu_j^k(\boldsymbol{x})\}.$$

In addition, we assume trapezoidal one-dimensional membership functions:

$$\mu_{j,i}^k(x_i) = \begin{cases} \frac{x_i-a}{b-a}, & a < x_i < b, \\ 1, & b \leqslant x_i \leqslant c, \\ \frac{d-x_i}{d-c}, & c < x_i < d, \\ 0, & x_i \leqslant a \vee x_i \leqslant d. \end{cases}$$

that is, each one-dimensional membership function is described through four parameters *a,b,c,d* and the corresponding fuzzy point can be written as follows

if x_i is$\langle a_i, b_i, c_i, d_i\rangle$ then y is k.

The open interval (a, d) thus describes the support area, whereas $[b, c]$ indicates its core.

In n dimensions a fuzzy point R belonging to region k is now expressed as:

$$\begin{array}{lll} \text{if} & x_1 \text{ is } \langle a_1, b_1, c_1, d_1\rangle & \\ & \quad\text{and}\ldots & \\ \text{and} & x_n \text{ is } \langle a_n, b_n, c_n, d_n\rangle & \\ \text{then} & \qquad\qquad y \text{ is } k & \end{array}$$

The algorithm presented in this section automatically constructs a fuzzy graph based on a set of examples. The partitioning of the input variables is determined automatically from the examples. The used algorithm is derived from a constructive neural network training algorithm [1]. In this paper a modified version of that algorithm, based on [9], is used to automatically find a set of soft rules that describe the training data.

Fig. 3 shows an example of a fuzzy graph constructed by the proposed algorithm.

The output variable Y is already partitioned into 6 regions (or classes) with user defined membership functions. The fuzzy graph consists of fuzzy points with individual membership functions for the input variable X which are determined during training.

3.2. Using soft targets

Construction of the fuzzy graph requires input patterns with a corresponding output. The output value (or *target*) can be defined as a soft value, using an individual membership function μ_{T}. For practical applications these soft targets can originate from measurements, for example sensor values with a certain accuracy. (Training patterns with sharp targets can be provided using a singleton as the corresponding output membership function.) From this soft target membership

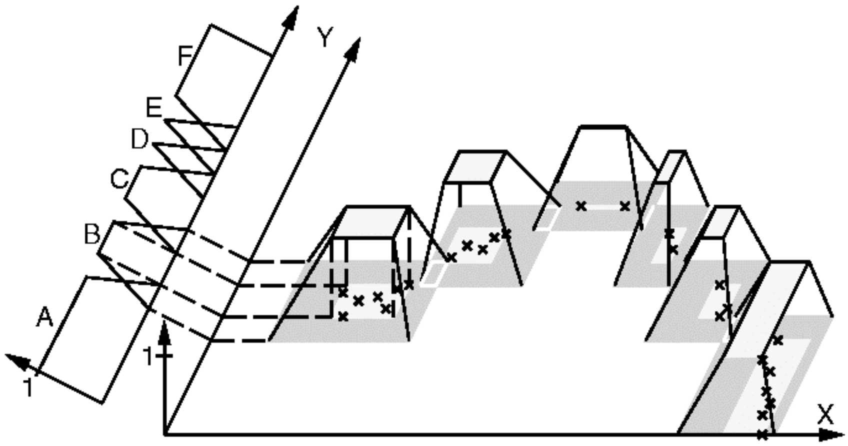

Fig. 3. An example for the type of fuzzy graphs considered in this paper. The output granularization is known a-priori, in this case six regions with individual membership functions are used. The presented algorithm then places fuzzy points in the input space to approximate the training examples (×).

μ_T values μ_k for each output region k are computed, using the pre-defined membership functions μ_k^o. Fig. 4 shows an example for the used fuzzification. The membership values for all regions of the underlying fuzzy graph are computed using a fuzzy and-operator (min) between the target and the set of output membership functions. Training is then performed using these target membership values. This leads to training patterns consisting of an input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ with the corresponding target $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_c)$ $(0 \leqslant \mu_i \leqslant 1)$, where c denotes the number of output regions.

The algorithm presented in this paper makes sure that each training pattern is covered by a fuzzy point of the region with the highest membership value and that fuzzy points of regions with membership values = 0 do not cover the pattern. This enables the fuzzy graph to tolerate moderately noisy patterns or small oscillations along boundaries, as will be demonstrated later. For the example target μ_T shown in Fig. 4 a fuzzy point of region C will cover the observed training

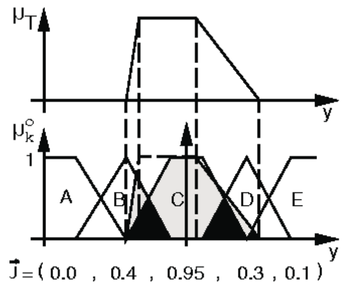

Fig. 4. Fuzzy inference is used to compute membership values to construct the fuzzy graph. A soft value represented through an individual membership function μ_T acts as target. Training is performed using the resulting membership values μ that are shown on the bottom.

pattern. Existing points belonging to B, D, and E are allowed to cover and points of A will be shrunk to avoid coverage, because $\mu_C > \mu_B,\ \mu_D,\ \mu_E > 0$ and $\mu_A = 0$.

3.3. The algorithm

The training algorithm is based on three steps that introduce new fuzzy points when necessary and adjust the core-and support-regions of existing ones [2]:

- **covered**: if a new training pattern lies inside the support-region of an already existing fuzzy point belonging to the output region with the maximum μ_k, the core-region of this fuzzy point is extended to cover the new pattern, which is-even in high-dimensional space-an easy task. Applet 1 (which can be found under 'Electronic annexes' on www.elsevier.com/locate/ida) demonstrates this.

  This procedure can be implemented through:
  if $\forall_i : 1 \leqslant l \leqslant n$: $x_i \in [a_i, d_i]$ then
  for $i = 1$ to n do
  $b_i' := \min\{b_i, x_i\}$
  $c_i' := \max\{c_i, x_i\}$
  endfor
- **Commit**: if a new pattern is not covered, a new fuzzy point corresponding to the region with maximum μ_k will be introduced. Its core-region is initialized using the new training instance with a non-existing support-region; that is, this fuzzy point covers the entire input space. Applet 1 illustrates this as well.
- **Shrink**: if a new pattern is incorrectly covered by an already existing fuzzy point of an incorrect region ($\mu_l = 0$), this fuzzy point's support-area will be reduced (e.g. shrunk) so that the conflict is solved. Applet 2 (which can be found under 'Electronic annexes' on www.elsevier.com/locate/ida) demonstrates this procedure.

Here the implementation is not as easy because reducing a rectangle (i.e. the support-region of the fuzzy point) in an n-dimensional space in a way to avoid overlap with a previously covered point has n possible alternatives (it is obvious that reducing the rectangles size along solely one dimension is enough to move the conflicting point outside of the resulting area). Using two heuristics is sufficient to reach satisfactory results in most cases:

- **large fuzzy points**: to avoid an unnecessary big shrink of a fuzzy point, its volume has to stay as large as possible. The volume of a fuzzy point is computed from its support: $V_{\text{rule}} = \Pi_{i=1\ldots n}(d_i - a_i)$. Finding the dimension, along which to shrink results in the fuzzy point with the maximum remaining volume (or the minimum loss in volume), can be computed through

$$\begin{aligned} i_{\max} &= \operatorname{argmax}_{i=1\ldots n}\{\max\{(x_i - a_i), (d_i - x_i)\} \bullet \Pi_{j=1\ldots n, j\neq i}(d_j - a_j)\} \\ &= \operatorname{argmin}_{i=1\ldots n}\{(x_i - a_i), (d_i - x_i)\}; \end{aligned}$$

- **balanced fuzzy points**: to avoid long, thin fuzzy points, a dimension can only be shrunk below a certain threshold ε when all other possible shrinks would result in a dimension below ε as well. This procedure can be implemented through:

[2] In the following the indices indicating the fuzzy point's index j and region k are omitted for sake of readability. The context will make clear which index and region is meant.

if $\forall_i : 1 \leqslant i \leqslant n : x_i \in [a_i, d_i]$ then
    // compute all possible shrinks
    for $i = 1$ to n do
        if $(d_i - x_i) > (x_i - a_i)$ then
            $a'_i := x_i$
            $d'_i := d_i$
        else
            $a'_i := a_i$
            $d'_i := x_i$
        endfor
    // choose best shrink (maximise remaining volume)
    if $\exists_i : (d'_i - a'_i) > \varepsilon$ then
        // if possible without violating ε-condition
        $j = \text{argmin}_{j=1\ldots n,(d'_j - a'_j)>\varepsilon}\{(d_j - a_j) - (d'_j - a'_j)\}$
    else
        // otherwise just minimize loss in volume
        $j = \text{argmin}_{j=1\ldots n}\{(d_j - a_j) - (d'_j - a'_j)\}$
    // shrink selected dimension
    $a_j = a'_j$
    $d_j = d'_j$
endif

The entire procedure for training of one complete epoch looks as follows:

        // reset weights and cores:
(1) FORALL fuzzy points R_i^k DO
            $A_i^k = 0.0$
            $\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \rangle_i^k = \langle \mathbf{a}, -, \text{-}, \mathbf{d} \rangle_i^k$
        ENDFOR
        // train one complete epoch
(2) FORALL training pattern $(\boldsymbol{x}, \boldsymbol{\mu})$ DO
            // determine maximum $\mu^{\mathrm{k}}(\boldsymbol{x})$
            $k = \text{argmax}_{1 \leqslant k \leqslant c}\{\mu^k(\boldsymbol{x})\}$
(3)   IF $\exists R_i^k : \boldsymbol{x} \in [a_i^k, d_i^k]$ THEN
(4) $A_i^k = A_i^k + 1$
                CALL **covered**
            ELSE
(5)   // "commit": introduce new fuzzy point
                $m_{\mathrm{k}} = m_{\mathrm{k}} + 1$
(6) $\langle \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \rangle_{mk}^k = \langle -, \mathbf{x}, \mathbf{x}\text{-} \rangle$
                $A_{mk}^k = 1$
            ENDIF
            // "shrink": adjust conflicting fuzzy points
(7)   FORALL fuzzy points R_j^1 and regions l with $\mu^1(\mathbf{x}) = 0$ DO
                IF $\mathbf{x} \in [a_j^1, d_j^1]$ THEN
                    CALL **shrink**
            ENDFOR
        ENDFOR

First, all weights A_i^k are set to zero to avoid accumulation of erroneous information about the training patterns (1). In addition the core region is erased, because the cores might shrink in subsequent epochs. Next all training patterns are presented (2). If the new pattern is already covered by a fuzzy point of the region with the highest μ_k (3), the weight of the largest covering fuzzy point is increased (4) and the core is set to cover this point. Otherwise a new fuzzy point is introduced (5), having the new pattern as its reference (6). Its initial size is set infinite. The last step must include shrinking all fuzzy points of conflicting regions l (e.g. $\mu_l = 0$) if their support-region covers this specific pattern (7). Due to the iterative nature of the algorithm, fuzzy points may be introduced that are too small and will be covered by larger fuzzy points later during training.

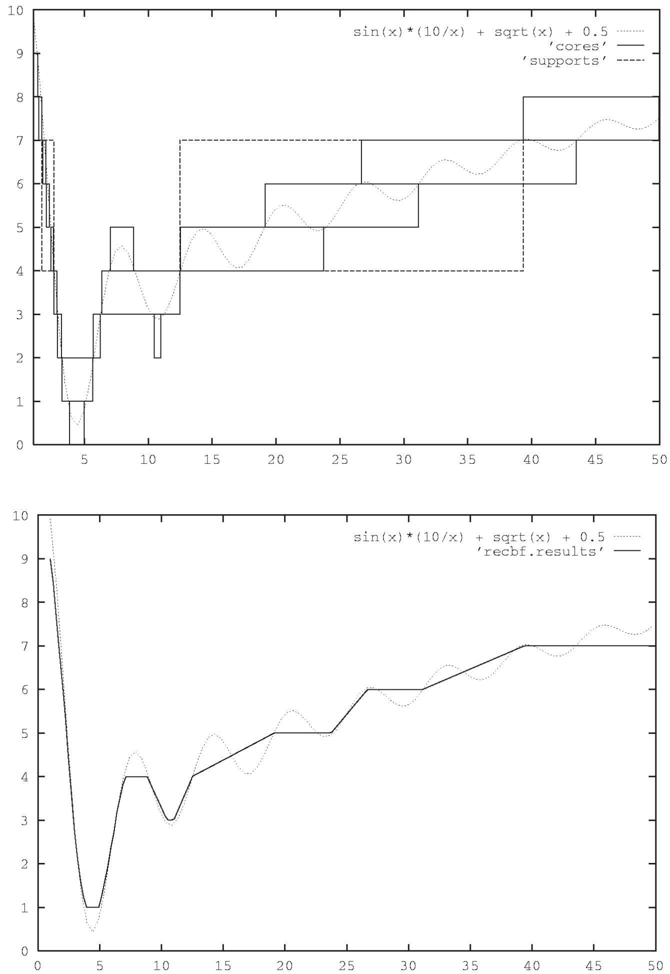

Fig. 5. A fuzzy graph approximates an artificial function. All core-regions but only the support-areas for the fuzzy points of region $\mu_5^o = \langle 4,5,6,7 \rangle$ are shown (top). The picture on the bottom shows the corresponding approximation.

These fuzzy points can be deleted afterwards, because their weight will remain zero during subsequent epochs. This last step is not shown in Fig. 5.

After termination of training, which usually takes only 4–6 epochs, the final set of fuzzy points forms a fuzzy graph. Each fuzzy point is associated with one output region and computes a membership value for a certain input pattern. The maximum degree of membership of all fuzzy points for one region determines the overall degree of membership. Fuzzy inference then produces a soft value for the output and using the well-known center-of-gravity method a final crisp output value can be obtained, if so desired.

The two applets 3 and 4 (which can be found under 'Electronic annexes' on www.elsevier.com/locate/ida) demonstrate the algorithm. Applet 3 shows how rules of two different regions are placed in a two-dimensional space, whereas applet 4 allows to construct a fuzzy graph from user-defined example points in the x-y-space.

4. Approximation results

To demonstrate the presented approach an artificial function was used. Fig. 5 shows the underlying, one-dimensional function and the constructed fuzzy graph after presenting 2000 randomly generated training patterns using an equidistant 10 region output-granularization. Note how fuzzy points cover regions of different widths and are only introduced where necessary. On the right the resulting function approximated by the fuzzy graph is shown.

For comparison we include results of two other algorithms [24] and [8]. The grid based approach with a fixed grid spacing [24] has problems finding a reasonable approximation as soon as the approximated function shows extrema that do not fit onto that grid, as can be seen in Fig. 6.

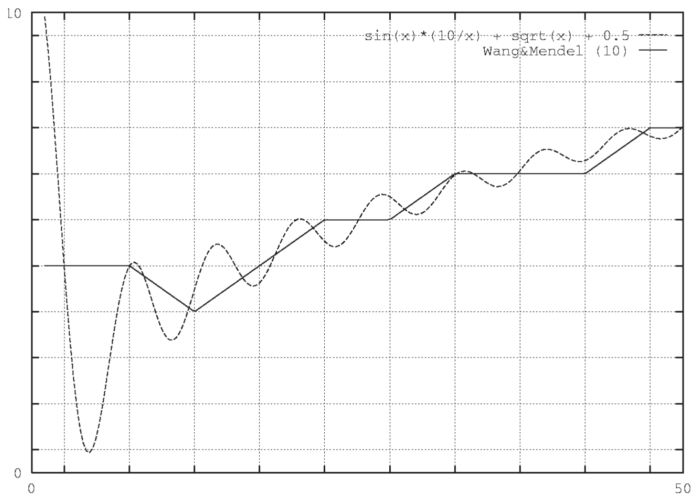

Fig. 6. An approximation of a fuzzy rule set generated by the algorithm of Wang &Mendel [24]. 10 classes were used for the input and output fuzzification.

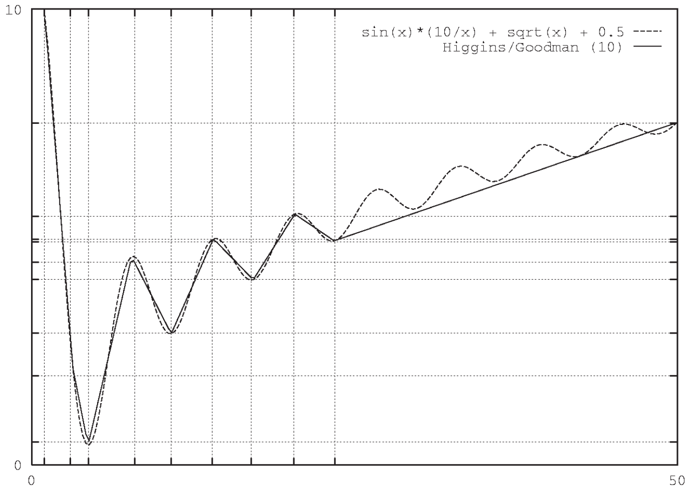

Fig. 7. An approximation of a fuzzy rule set generated by the algorithm of Higgins & Goodman [8]. The algorithm was run until a maximum of 10 classes was used for the input and output fuzzification.

Here some peaks of the function lie far from existing rule centers and as a result do not influence the resulting rule set.

The other algorithm [8] uses a flexible partitioning, by introducing new membership functions at points of maximum error. Allowing a maximum of 10 membership functions for each variable results in an approximation as shown in Fig. 7. This methodology, however, is extremely sensitive towards outliers in the training data, because new rules are placed at points of maximum error which are often caused by outliers. Especially in the case of noisy datasets this may cause problems.

For many tasks it is of key interest, how well a certain amount of noise can be handled. A series of experiments were conducted with a specific amount of noise added to the training data. This was done to demonstrate the effect of noisy training patterns on the generation of fuzzy graphs by the proposed algorithm. Fig. 8 shows two examples. On the top equally distributed noise (± 0.5) was added to the output value, which is handled very well by the fuzzy graph. On the right the same training data with noise four times bigger (± 2.0) was used. This time the fuzzy graph fails to tolerate the noise and tries to model the training points with large error resulting in an increasing number of required fuzzy points. The error on the underlying noise-free function stays within a certain range, until the noise reaches a specific level. After this point the fuzzy graph starts to model the noisy data points, resulting in an increasing error rate on the test data. An additional indicator for this effect is the exploding number of fuzzy points required to fit the fuzzy graph to the training points.

This example shows that the resulting fuzzy graph approximates the original function well, with a certain degree of accuracy. The amount of smoothing can be controlled by the output fuzzification. Using more and finer membership functions results in higher precision. In regions containing "noise" up to a certain amount, the fuzzy graph ignores the oscillations and tends to produce plateaus.

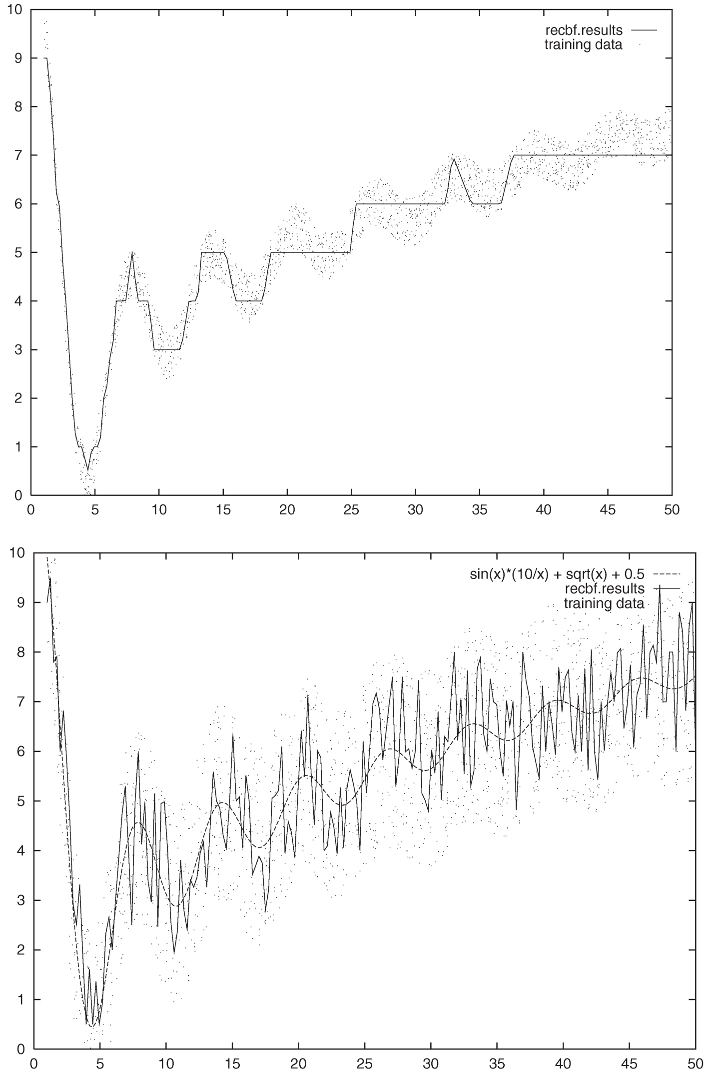

Fig. 8. Influence of noisy training patterns on the construction of the fuzzy graph. Training data on the top included ± 0.5 equally distributed noise, which the fuzzy graph was able to tolerate. The picture on the right shows how the fuzzy graph tries to model the noise if it becomes too large (± 2.0).

5. Fuzzy graphs as metamodels

One application where the use of fuzzy graphs can be extremely beneficial is the automatic construction of models from real systems. In the Section 5.1 we will demonstrate how the proposed methodology can be used to build an auxiliary model (or *metamodel*, other approaches are discussed in Refs. [4,10,14]) from data generated by simulation experiments with a real-world token bus model.

The used token bus system belongs to the class of field bus systems, that is, a special type of communications systems, designed to connect machines and computers in a manufacturing environment. In this section the analysis will mainly focus on the real time facility of the model, that is, its capability to respond to each request within a limited time. To guarantee this property for the given simulation model a fuzzy graph was built using the presented method and its behaviour depending on different parameter settings was explored.

The modelled token bus system corresponds to the seven level architecture of the ISO/OSI communication standard. Fig. 9 shows the structure of the system.

Many details like different message priorities and the token handling had to be taken into account when modelling the system with a queuing network model. The model was then implemented with a commercially available simulation environment. To illustrate the complexity of the underlying queuing network the internal structure of one module with its interfaces is depicted on the right-hand side of Fig. 10. Since each station is modelled by four different modules the whole model consists of more than two hundred different queues and several hundreds connections. The complexity of the internal structure causes a conventional analysis of this model to be extremely time-consuming and complicated.

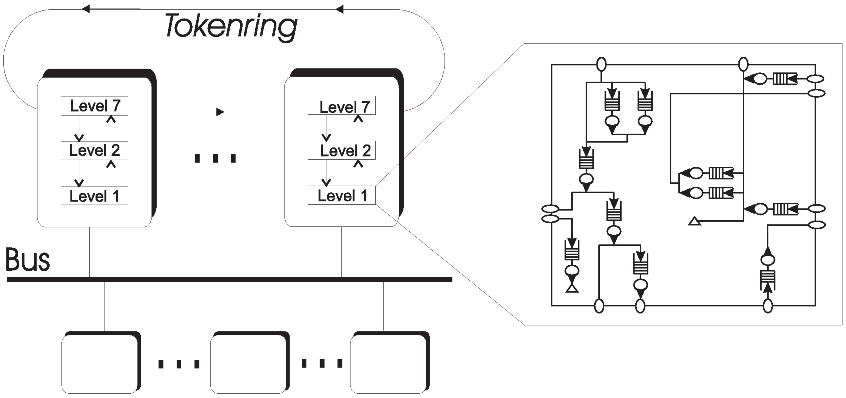

Fig. 9. The model.

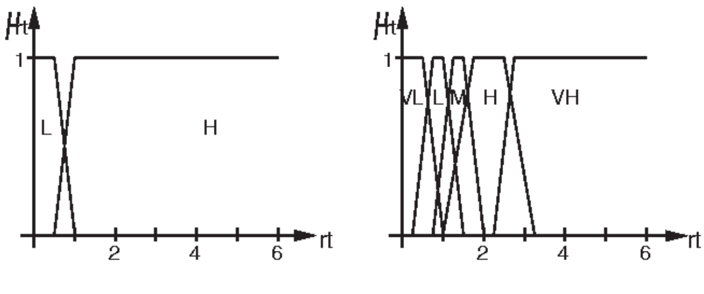

Fig. 10. Two used types of membership functions.

5.1. Data generation and analysis

Due to the large number of parameters (20 input and 10 output parameters) of the complete model the example analysis presented in this section will focus on one output parameter and a subset of the available inputs. The output of interest is the response time between two master stations. It is desirable that this response time always stays below an upper bound to guarantee that the reaction of the system is always in time. The four following input parameters were chosen while the remaining parameters remained fixed:

- average time for execution (**cpu1**): describes the performance of the CPU module of station 1, i.e., the average time required to execute one command. This value is varied within 0.1 (fast) and 3.4 (slow).
- workload rate (**workload**): describes the average idle time between two requests, this value is varied within 0.02 (low idle time, high workload) and 1.0 (low workload).
- target-rotation-time (**trt**): maximum allowed time to process the token. This parameter controls the time each station has to send messages, values were set within [0.01, 0.4].
- number of additional stations (**stations**): represents the background workload on the network. Many additional stations communicating over the network will increase the traffic on the network. The number of stations was varied in Refs. [1,15].

Since the construction of the metamodel only depends on the example data these examples have to be representative. For this the planning of simulation experiments must be done carefully. In our application a full factorial design is not possible, therefore we used randomised settings for the input parameters. 350 simulation experiments were performed where the input parameter values were varied randomly within the given intervals. The averaged response time (**rt**) was measured within (0.088, 9.75). Each simulation experiment was repeated five times with a different random number stream of the simulation tool. From these five values the minimum, the maximum, and the average were taken and a triangular target membership function was generated.

With the fuzzy graph approach these four-dimensional data-vectors with their corresponding target membership functions were used for training. Since the main focus of attention were fast responses (i.e. low values of **rt**) the membership functions for low values are defined finer than those for bigger values (Fig. 10). Three series of experiments were performed with two, five and ten membership functions.

Construction of the fuzzy graph required about 10 seconds on a SUN Sparc10 workstation. No training parameters besides the a priori definition of the output membership functions had to be considered or tuned. While a simulation run takes about 200 seconds the propagation of a new parameter set through the fuzzy graph is completed within fractions of a second, resulting in an increase in speed of two orders of magnitude. As expected the metamodel can be used for much faster simulation.

To judge the reliability of the complete fuzzy graph metamodel the quality can be analysed by computing the mean relative error of the approximation of the metamodel. For this analysis an independent data set that was not used for building the fuzzy graph, the so-called cross-validation set, was used. The data set of 350 vectors was split into one tenth for testing and nine tenths for building the graph and using each tenth once for testing ten cross-validation runs were performed. The average error on the corresponding test data was 4.4% ± 1.0% (avg. 32 fuzzy points) with two, 4.1% ± 1.2% (avg. 59 fuzzy points) with five, and 3.3% ± 0.9% (avg. 68 fuzzy points) with ten membership functions for the output. This approximation quality is sufficient because the primary goal of the presented approach is the extraction of few understandable rules instead of achieving minimal approximation errors.

One of the resulting rule bases from an experiment with two output regions was used for further analysis. In this case the output regions are labelled *low* (**L**) and *high* (**H**). Since the main focus of analysis were parameter settings, which result in a low response time, rules of region *low* were investigated. From 29 rules 16 belong to this region and according to the rule weight the most important rule was

$$\begin{array}{llll} \text{if} & \textbf{cpu1} \text{ is} & & \langle -, 0.11, 1.69, 1.70 \rangle \\ \quad \text{and} & \textbf{workload} \text{ is} & & \langle -, 0.03, 0.99, - \rangle \\ \quad \text{and} & \textbf{trt} \text{ is} & & \langle -, 0.01, 0.39, - \rangle \\ \quad \text{and} & \textbf{stations} \text{ is} & & \langle 4, 5, 15, - \rangle \\ \text{then} & \textbf{rt} \text{ is} & \text{low} = & \langle -, 0.0, 0.5, 1.0 \rangle \\ & (\text{weight}: \ 116) & & \end{array}$$

This rule demonstrates how the core always covers a confident subset of the support-region. Here, for two parameters, namely **workload** and **trt** (target-rotation-time), the core covers the whole range of these parameters. Additionally it is only limited into one direction on the other two parameters, indicated by a support region having finite boundaries. The performing time of **cpu1** has to be below 1.70 and the number of additional stations above 4. This indicates that a certain amount of computation power together with some background stations guarantees fast responses no matter what settings are chosen for **workload** and performing time **trt**. In addition the weight of this rule can be used to judge its reliability. The weight indicates the number of examples that are covered by this rule. In the rule shown above 116 examples fall inside its core region. This means that about 36% of all examples are covered by this rule, indicating a high reliability. Rules with low weight on the other hand might be indicators for outliers, irregularities in the data set or regions of high sensitivity, i.e. regions where small changes of the attributes result in large variations of the output.

Another question of interest is the influence of some parameters considering the output. In this example the above rule indicates that the target-rotation-time **trt** has no influence on the response time if the CPU is fast (below 1.7) and at least 5 background stations exist because the support of **trt** is not restricted and the core covers nearly the whole domain of this parameter.

It can also be of interest to find "bad" examples, i.e. regions where the response time is very high. These indicate parameter settings that should be avoided. For example, the rule with the highest weight for response time **rt** = high was:

$$\begin{array}{llll} \text{if} & \textbf{cpu1} \text{ is} & & \langle 2.70, 2.71, 3.39, - \rangle \\ \quad \text{and} & \textbf{workload} \text{ is} & & \langle -, 0.09, 0.98, - \rangle \\ \quad \text{and} & \textbf{trt} \text{ is} & & \langle 0.13, 0.14, 0.39, - \rangle \\ \quad \text{and} & \textbf{stations} \text{ is} & & \langle -, 1, 15, - \rangle \\ \text{then} & \textbf{rt} \text{ is} & \text{high} = & \langle 0.5, 1.0, 1.0, - \rangle \\ & (\text{weight}: \ 38) & & \end{array}$$

This rule indicates that if the CPU is very slow and the target-rotation-time **trt** is above a certain value the response time is high no matter what workload is considered (represented by background stations and the time between requests). Therefore if the system includes a slow CPU module the target-rotation-time should be set carefully. Since only 85 examples are of region *high* the weight of 38 is an indication for a high reliability of this rule.

These results illustrate the applicability of the presented approach for metamodelling tasks. The approximation error indicates the reliability of the metamodel and the fuzzy graph can be used for new simulation experiments. In addition the example rules deliver helpful information about dependencies between factors and the output of interest.

6. Conclusion

In this paper an approach to function approximation using an algorithm that automatically builds a fuzzy graph has been proposed. Construction of the fuzzy graph is done in a fast and efficient way without any need for user interaction, besides definition of the output granularization, which enables the user to define foci of attention. Evaluation of new data points is straightforward and the resulting representation is easy to understand. In addition it is possible to use soft targets for training, that is, fuzzy numbers or noisy values. The usage of fuzzy graphs to analyze complex simulation models was demonstrated on a real world token bus example. The presented approach was well suited to quickly and automatically extract an understandable metamodel simply from observations of the model. Therefore the presented approach offers an efficient way to build approximators from training data. In addition the fuzzy graph can be seen as a collection of fuzzy rules, thus offering an understandable representation of the acquired knowledge.

Acknowledgements

The authors would like to thank Prof. D. Schmid for his support and the opportunity to work on this interesting project. Thanks also to the reviewers for their positive and very helpful feedback. M. Berthold was supported by a stipend of the ‘‘Gemeinsame Hochschulsonderprogramm III von Bund und Laendern’’ through the DAAD.

References

[1] M.R. Berthold, J. Diamond, Constructive training of probabilistic neural networks, Neurocomputing 19 (1998) 167–183.
[2] M.R. Berthold, K.-P. Huber, Building fuzzy graphs from examples, IEEE International Conference on Fuzzy Systems 1 (1996) 608–613.
[3] M.R. Berthold, K.-P. Huber, Missing values and learning of fuzzy rules, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6 (2) (1998) 171–178.
[4] R.W. Blanning, The construction and implementation of metamodels, Simulation 24 (1975) 177–184.
[5] P. Clark, T. Niblett, The CN2 induction algorithm, Machine Learning 3 (1989) 261–283.
[6] M. Delgado, A. Gonzalez, An inductive learning procedure to identify fuzzy systems, Fuzzy Sets and Systems 55 (1993) 121–132.
[7] S.K. Halamuge, M. Glesner, FuNe Deluxe: A group of fuzzy-neural methods for complex data analysis problems, in: Proceedings of the EUFIT’95, 1995.
[8] C.M. Higgins, R.M. Goodman, Learning fuzzy rule-based neural networks for control, in: Advances in Neural Information Processing Systems, vol. 5, pp. 350–357, Morgan Kaufmann, Los Altos, 1993.
[9] K.-P. Huber, M,R. Berthold, Building precise classifiers with automatic rule extraction, IEEE International Conference on Neural Networks 3 (1995) 1263–1268.

[10] J.P.C. Kleijnen, Model behaviour: Regression metamodel summarization, Encyclopedia of Systems and Control 5 (1987) 3024–3030.
[11] X. Liu, P. Cohen, M.R. Berthold, Advances in Intelligent Data Analysis-Reasoning about Data LNCS1280, Springer, Berlin, 1997.
[12] C. McMillan, M.C. Mozer, P. Smolensky, Rule induction through integrated symbolic and subsymbolic processing, in: Advances in Neural Information Processing Systems, vol. 4, pp. 969–976, Morgan Kaufmann, 1992.
[13] D. Nauck, R. Kruse, Nefclass-a neuro-fuzzy approach for the classification of data, in: Proceedings of the Symposium on Applied Computing, 1995.
[14] H. Pierreval, Rule-based simulation metamodels, European Journal of Operational Research 61 (1992) 6–17.
[15] J.R. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.
[16] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Los Altos, 1993.
[17] S. Salzberg, A nearest hyperrectangle learning method, Machine Learning 6 (1991) 251–276.
[18] P.K. Simpson, Fuzzy min–max neural networks-part 1: classification, IEEE Transactions on Neural Networks 3 (5) (1992) 776–786.
[19] P.K. Simpson, Fuzzy min–max neural networks- part 2: clustering, IEEE Transactions on Fuzzy Systems 1 (1) (1993) 32–45.
[20] T. Takagi, M. Sugeno, Fuzzy identification of systems and its appications to modeling and control, IEEE Transactions on Systems, Man, and Cybernetics 15 (1) (1985) 116–132.
[21] S. Thrun, Extracting rules from artificial neural networks with distributed representations, in: Advances in Neural Information Processing Systems, vol. 7, pp. 505–512, MIT Press, Cambridge, 1995.
[22] G. Towell, J.W. Shavlik, Interpretation of artificial neural networks: mapping knowledge-based neural networks into rules, in: Advances in Neural Information Processing Systems, 4, pp. 977-984, Morgan Kaufmann, Los Altos, 1992.
[23] V. Uebele, S. Abe, M.-S. Lan, A neural-network-based fuzzy classifier, IEEE Transactions on Systems Man and Cybernetics 25 (2) 1995.
[24] L.-X. Wang, J.M. Mendel, International Symposium on Intelligent Control, IEEE Press, New York, 1991, pp. 263–268.
[25] D. Wettschereck, A hybrid nearest-neighbour and nearest-hyperrectangle learning algorithm, Proceedings of the European Conference on Machine Learning (1994) 323–335.
[26] L.A. Zadeh, Soft computing and fuzzy logic, IEEE Software (1994) 48–56.
[27] L.A. Zadeh, Fuzzy logic and the calculi of fuzzy rules and fuzzy graphs: A precis, Multi. Val. Logic 1 (1996) 1–38.