



Constructing Fuzzy Graphs from Examples

Michael R. Berthold* and Klaus-Peter Huber†

TR-97-053

December 1997

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.

*Michael Berthold is with ICSI and the University of California at Berkeley, Computer Science Division, Department of EECS, Berkeley, CA 94720, USA. E-mail: berthold@cs.berkeley.edu

†Klaus-Peter Huber is with the University of Karlsruhe, IRF (Prof. D. Schmid), Am Zirkel 2, 76128 Karlsruhe, Germany. E-mail: KlausPeter.Huber@Informatik.Uni-Karlsruhe.DE

I. 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 behavior. 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.

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 [1], [2], [3], or for a method that extracts fuzzy rules [4]). Extracting rules directly from the data is usually complicated, especially in the case of noisy data examples, due to the crisp nature of the underlying rules [5], [6]. Decision trees offer an alternative, but are usually harder to understand [7], [8], [9].

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 [10]). Finding automatically 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 [11] 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 [12] 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 [13]). Also tree based approaches for fuzzy rules have been proposed [14]. In most cases, however, it is desired to place rules only in regions of interest. In [15] 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 [16] points out that using a non grid oriented set of rules leads to a relatively small number of fuzzy rules to characterize a complex relationship between two or more variables. An approach that tries to find individual fuzzy hyperboxes via a growth process was proposed in [17], [18] 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.

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 form of a fuzzy graph. The used algorithm is derived from a constructive neural network training method [19] 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 behavior.

II. FUZZY GRAPHS

Zadeh illustrates fuzzy graphs in [20] 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 (Figure 1, see [20] for more details). 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 ($1 \leq i \leq n$).

The typical fuzzy graphs build upon *fuzzy points* of the type

$$\text{if } x_1 \text{ is } A_1 \text{ and } \dots x_n \text{ is } A_n \text{ then } y \text{ is } B$$

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

$$\text{if } \vec{x} \text{ is } \mathbf{A} \text{ then } y \text{ is } B$$

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

$$(\vec{x}, y) \text{ is } \mathbf{A} \times B$$

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

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

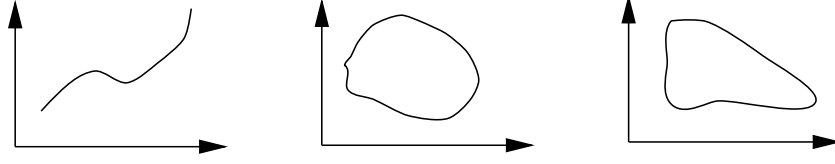


Fig. 1. Approximate representations of functions, contours, and relations (after Zadeh [20]).

or, more precisely

$$\begin{aligned} \mu_{\mathbf{A} \times \mathbf{B}}(\vec{x}, y) &= \\ &= \mu_{A_1}(x_1) \wedge \cdots \wedge \mu_{A_n}(x_n) \wedge \mu_B(y) \\ &= \min \{ \mu_{A_1}(x_1), \dots, \mu_{A_n}(x_n), \mu_B(y) \} \end{aligned}$$

Zadeh [16] uses the term *fuzzy point* to denote $\mathbf{A} \times \mathbf{B}$. A collection of such rules can now be regarded as forming a superposition of n fuzzy points:

$$(\vec{x}, y) \text{ is } (\mathbf{A}_1 \times B_1 + \cdots + \mathbf{A}_m \times B_m)$$

where $+$ denotes the disjunction operator (usually defined as maximum). Zadeh calls this characterization of a dependency *fuzzy graph*, because the collection of rules can be seen as a coarse representation of a functional dependency f^* of y on \vec{x} . This fuzzy graph f^* can thus be defined as:

$$f^* = \sum_{j=1}^m (\mathbf{A}_j \times B_j)$$

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 \mathbf{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 [11], [12]).

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

$$\frac{\vec{x} \text{ is } \mathbf{A}}{f^* \text{ is } \sum_{j=1}^m \mathbf{A}_j \times B_j} \quad \underline{\hspace{10em}} \quad \frac{\hspace{10em}}{y \text{ is } B}$$

results in an intersection of the fuzzy graph f^* with a cylindrical extension of the input fuzzy set \mathbf{A} . Figure 2 shows an example. This functional dependency can be computed through:

$$\begin{aligned} \mu_{B=f^*(\mathbf{A})}(y) &= \sup_{\vec{x}} \{ \mu_{f^*}(\vec{x}, y) \wedge \mu_{\mathbf{A}}(\vec{x}) \} \\ &= \sup_{\vec{x}} \{ (\mu_{\mathbf{A}_1 \times B_1}(\vec{x}, y) \vee \cdots \vee \mu_{\mathbf{A}_m \times B_m}(\vec{x}, y)) \\ &\quad \wedge \mu_{\mathbf{A}}(\vec{x}) \} \end{aligned}$$

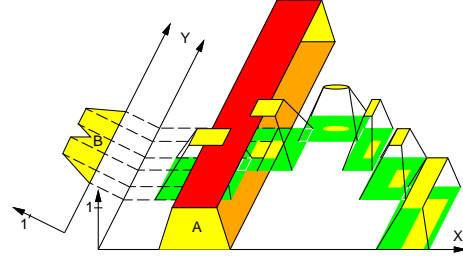


Fig. 2. Interpolation as the intersection of a fuzzy graph with a cylindrical extension of A .

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

III. AUTOMATIC CONSTRUCTION OF FUZZY GRAPHS

The algorithm presented here constructs a fuzzy graph based on example data points (\vec{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 $1 \leq k \leq c$.

A. Trapezoidal Fuzzy Graphs

The resulting fuzzy graph consists of a set of fuzzy points or fuzzy rules R_j^k , $1 \leq j \leq m_k$, where m_k indicates the number of rules for region k . If the input space is of dimensionality n , each input vector \vec{x} consists of n attributes: $\vec{x} = (x_1, \dots, x_n)$, $x_i \in \mathcal{R}$ (\mathcal{R} denotes the domain of the attributes, usually the real numbers). A rule’s activity $\mu_j^k(\vec{x})$ indicates the degree of membership of the pattern \vec{x} to the corresponding rule R_j^k . Using the normal notation of fuzzy rules, each rule can be decomposed into n individual, one-dimensional membership functions:

$$\mu_j^k(\vec{x}) = \min_{i=1, \dots, n} \{ \mu_{j,i}^k(x_i) \}$$

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

$$\mu^k(\vec{x}) = \max_{1 \leq j \leq m_k} \{\mu_j^k(\vec{x})\}$$

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

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

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:

if x_1 is $\langle a_1, b_1, c_1, d_1 \rangle$ and \dots
 x_n is $\langle a_n, b_n, c_n, d_n \rangle$ then y is k

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 [19]. In this paper a modified version of that algorithm, based on [21] is used to automatically find a set of soft rules that describe the training data.

Figure 3 shows an example of a fuzzy graph constructed by the proposed algorithm. The output

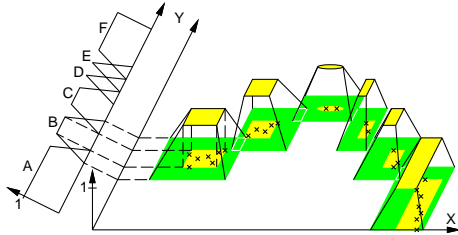


Fig. 3. An example for the type of fuzzy graphs considered in this paper. The output granularization is known a-priori, in this case 6 regions with individual membership functions are used. The presented algorithm then places fuzzy points in the input space to approximate the training examples (\times).

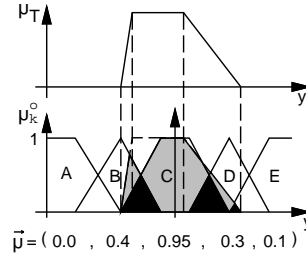


Fig. 4. Fuzzy inference is used to compute membership values to construct the fuzzy graph. A soft value represented through an individual membership function μ_T acts as target. Training is performed using the resulting membership values $\vec{\mu}$ that are shown on the bottom.

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.

B. 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 accuracy¹. From this soft target membership μ_T values μ_k for each output region k ($1 \leq k \leq c$) are computed, using the pre-defined membership functions μ_k^o . Figure 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 $\vec{x} = (x_1, \dots, x_n)$ with the corresponding target $\vec{\mu} = (\mu_1, \dots, \mu_c)$ ($0 \leq \mu_i \leq 1$), 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 Figure 4 a fuzzy point of region C will cover the observed training pattern, existing points belonging to B, D, and E are allowed to cover

¹Training patterns with sharp targets can be provided using a singleton as the corresponding output membership function.

and points of \mathbf{A} will be shrunk to avoid coverage, because $\mu_C > \mu_B, \mu_D, \mu_E > 0$ and $\mu_A = 0$.

C. The Algorithm

The used training algorithm is based on three steps that introduce new fuzzy points when necessary and adjust the core- and support-regions of existing ones²:

- **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. Figure 5 shows an example. This procedure can be implemented through:

```

IF  $\forall i : 1 \leq i \leq n : x_i \in [a_i, d_i]$  THEN
  FORALL  $i : 1 \leq i \leq n$  DO
     $b'_i := \min\{b_i, x_i\}$ 
     $c'_i := \max\{x_i, c_i\}$ 
  ENDFOR

```

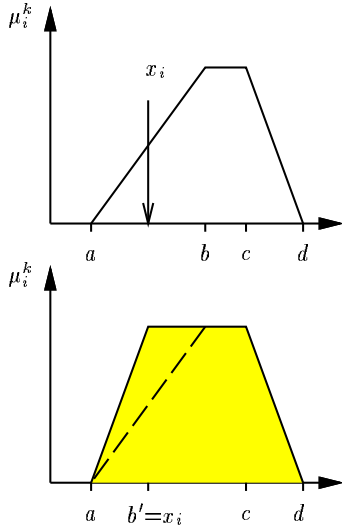


Fig. 5. Covered: An already existing fuzzy point $\langle a, b, c, d \rangle$ covers the new pattern. The new extended core-region $[x_i, c]$ includes the new pattern x_i .

- **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 assigned the new training instance with a non-existing support-region; that is,

²In 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 are meant.

this fuzzy point covers the entire input space. Figure 6 illustrates this step in one dimension.

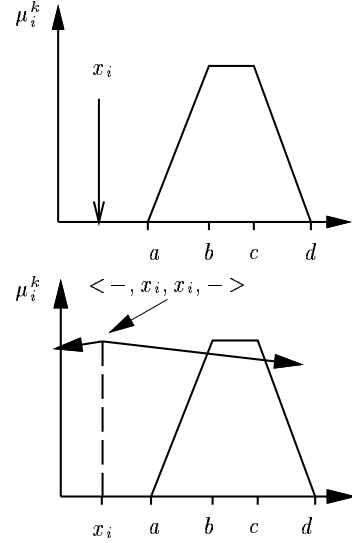


Fig. 6. Commit: A new fuzzy point $\langle -, x_i, x_i, - \rangle$ is introduced to cover the new pattern. Its support-region is non-existing and thus covers the entire input space.

- **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. Figure 7 illustrates 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:

1. 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:

$$V_{\text{rule}} = \prod_{i=1}^n (d_i - a_i)$$

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:

$$i_{\text{best}} = \underset{1 \leq i \leq n}{\operatorname{argmin}} \{ \min \{ (x_i - a_i), (d_i - x_i) \} \}$$

2. 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.

The complete procedure can be implemented through:

```

IF  $\forall i : 1 \leq i \leq n : x_i \in [a_j, d_j]$  THEN
  // compute all possible shrinks
  FORALL  $i : 1 \leq i \leq n$  DO
    IF  $(d_i - x_i) > (x_i - a_i)$  THEN
       $a'_i = x_i$ 
       $d'_i = d_i$ 
    ELSE
       $a'_i = a_i$ 
       $d'_i = x_i$ 
    ENDIFOR
  // choose best shrink (max remaining vol.)
  IF  $\exists i : (d'_i - a'_i) > \epsilon$  THEN
     $j = \arg \min_{\substack{1 \leq j \leq n \\ (d'_j - a'_j) > \epsilon}} \{(d_j - a_j) - (d'_j - a'_j)\}$ 
  ELSE
     $j = \operatorname{argmin}_{1 \leq j \leq n} \{(d_j - a_j) - (d'_j - a'_j)\}$ 
  // shrink selected dimension
   $a_j = a'_j$ 
   $d_j = d'_j$ 

```

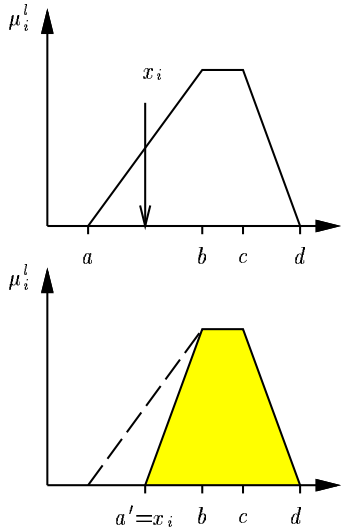


Fig. 7. Shrink: An already existing fuzzy point $\langle a, b, c, d \rangle$ belonging to a region l with $\mu_l^o(x_i) = 0$ covers the new pattern. Its support-region $[a, d]$ is shrunk to avoid future coverage of the presented pattern.

The entire procedure for training of one complete epoch is shown in Figure 8. First, all weights A_i^k are set to zero to avoid accumulation of erroneous information about the training patterns. In addition the core region is erased, because the cores might shrink in subsequent epochs.

Next all training patterns are presented. If the new pattern is already covered by a fuzzy point of the region with the highest μ_k , the weight of the largest covering fuzzy point is increased and the core is set to cover this point; otherwise a new fuzzy point is introduced, having the new pattern as its reference. 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. 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. These fuzzy points can be deleted afterwards, because their weight will remain zero during subsequent epochs. This last step is not shown in figure 8.

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

```

// reset weights and cores:
FORALL fuzzy points  $R_i^k$  DO
   $A_i^k = 0.0$ 
   $\langle \vec{a}, \vec{b}, \vec{c}, \vec{d} \rangle_i^k = \langle \vec{a}, -, -, \vec{d} \rangle_i^k$ 
ENDIFOR
// train one complete epoch
FORALL training pattern  $(\vec{x}, \vec{\mu})$  DO
  // determine maximum  $\mu_k(\vec{x})$ 
   $k = \arg \max_{1 \leq k \leq c} \{\mu_k(\vec{x})\}$ 
  IF  $\exists R_i^k : \vec{x} \in [a_i^k, d_i^k]$  THEN
     $A_i^k = A_i^k + 1$ 
    CALL covered
  ELSE
    // "commit": introduce new fuzzy point
     $m_k = m_k + 1$ 
     $\langle \vec{a}, \vec{b}, \vec{c}, \vec{d} \rangle_{m_k}^k = \langle -, \vec{x}, \vec{x}, - \rangle$ 
     $A_{m_k}^k = 1$ 
  ENDIF
  // "shrink": adjust conflicting fuzzy points
  FORALL  $l : \mu^l(\vec{x}) = 0 \wedge 1 \leq j \leq m_l$  DO
    IF  $\vec{x} \in [a_j^l, d_j^l]$  THEN
      CALL shrink
    ENDIFOR
  ENDIFOR

```

Fig. 8. The training algorithm for one epoch.

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.

IV. APPROXIMATION RESULTS

To demonstrate the presented approach an artificial function was used. Figure 9 (left) 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 [11] and [12]. The grid based approach with a fixed grid spacing [11] has problems finding a reasonable approximation as soon as the approx-

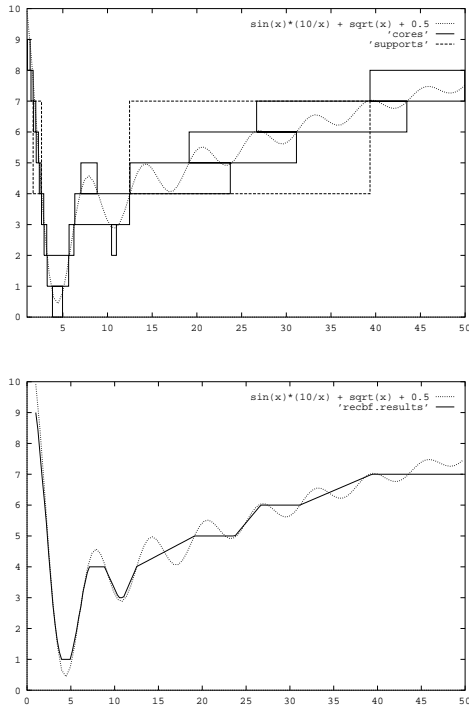


Fig. 9. A fuzzy graph approximates an artificial function. All core-regions but only the support-areas for the fuzzy points of region $\mu_5 = \langle 4, 5, 6, 7 \rangle$ are shown (left). The picture on the bottom shows the corresponding approximation.

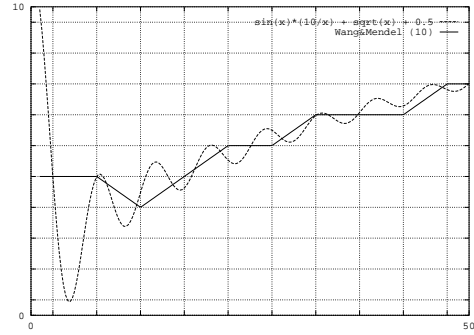


Fig. 10. An approximation of a fuzzy rule set generated by the algorithm of Wang&Mendel [11]. 10 classes were used for the input and output fuzzification.

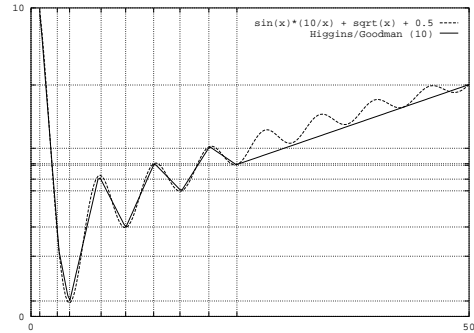


Fig. 11. An approximation of a fuzzy rule set generated by the algorithm of Higgins&Goodman [12]. The algorithm was run until a maximum of 10 classes was used for the input and output fuzzification.

imated function shows extrema that do not fit onto that grid, as can be seen in figure 10. 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 [12] 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 figure 11. 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 was conducted with a specific amount

of noise added to the training data, to demonstrate the effect of noisy training patterns on the generation of fuzzy graphs by the proposed algorithm. Figure 12 shows two examples. On the left 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

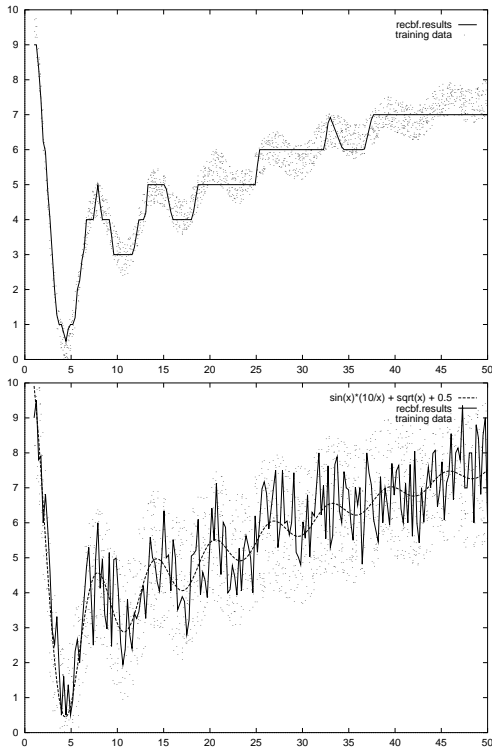


Fig. 12. Influence of noisy training patterns on the construction of the fuzzy graph. Training data on the top included ± 0.5 equally distributed noise, which the network was able to tolerate. The picture on the bottom shows how the network tries to model the noise if it becomes too large (± 2.0).

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.

V. FUZZY GRAPHS AS METAMODELLS

One application where the use of fuzzy graphs can be extremely beneficial is the automatic construction of models from real systems. In the following section we will demonstrate how the proposed methodology can be used to build an auxiliary model (or *meta-model*, other approaches are discussed in [22], [23], [24]) 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 behavior depending on different parameter settings was explored.

The modeled token bus system corresponds to the seven level architecture of the ISO/OSI communication standard. Figure 13 shows the structure of the system. Many details like different message priorities and the token handling had to be taken into account when modeling 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 side of Figure 13. Since each station is modeled 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.

A. 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 the response time between two master stations in dependence of a selection of parameters of interest. 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. Four

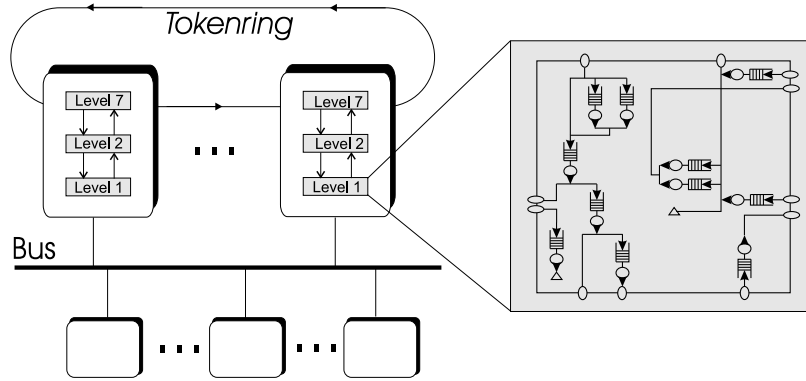


Fig. 13. The Model

input parameters were chosen while the other 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) — 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) — 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: [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 randomized 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 (Figure 14). Three series of experiments were performed with two, five and ten membership functions. Fuzzy graph construction 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 analyzed by computing the mean relative error of the approximation of the metamodel. For this analysis an independent dataset that was not used for building the fuzzy graph, the so called cross-validation set, was used. The dataset of 350 vectors was split into one tenth for testing and nine tenth for building the graph and using each tenth once for testing ten cross-validation runs were performed. The average error on the corresponding testdata was $4.4\% \pm 1.0\%$ (avg. 32 fuzzy points) with two, $4.1\% \pm 1.2\%$ (avg. 59 fuzzy points) with five, and $3.3\% \pm 0.9\%$ (avg. 68 fuzzy points) with ten membership functions for the output. This approximation quality is sufficient be-

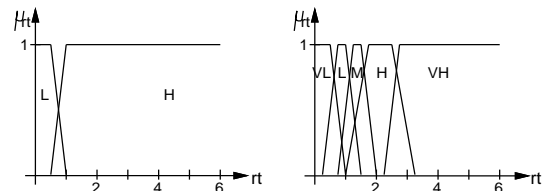


Fig. 14. Two Used Types of Membership Functions

cause 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 labeled *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:

```

IF      cpu1      IS < -, 0.11, 1.69, 1.70 >
AND    workload IS < -, 0.03, 0.99, - >
AND    trt       IS < -, 0.01, 0.39, - >
AND    stations IS < 4, 5, 15, - >
THEN   rt        IS low < -, 0.0, 0.5, 1.0 >
(weight: 116)

```

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 dataset 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 rule above indicates that the *target-rotation-time* 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:

```

IF      cpu1      IS < 2.70, 2.71, 3.39, - >
AND    workload IS < -, 0.09, 0.98, - >
AND    trt       IS < 0.13, 0.14, 0.39, - >
AND    stations IS < -, 1, 15, - >
THEN   rt        IS high < 0.5, 1.0, 1.0, - >
(weight: 38)

```

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 also of this rule.

These results illustrate the applicability of the presented approach for metamodeling 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.

VI. CONCLUSIONS

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.

ACKNOWLEDGMENTS

The authors would like to thank Prof. D. Schmid for his support and the opportunity to work on this interesting project. M. Berthold was supported by a stipend of the “Gemeinsame Hochschulsonderprogramm III von Bund und Ländern” through the DAAD.

REFERENCES

- [1] Sebastian Thrun, “Extracting rules from artificial neural networks with distributed representations”, in *Advances in*

- Neural Information Processing Systems*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds., Cambridge MA, 1995, 7, pp. 505–512, MIT Press.
- [2] G. Towell and J. W. Shavlik, "Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules", in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. 1992, pp. 977–984, Morgan Kaufmann.
 - [3] Clayton McMillan, Michael C. Mozer, and Paul Smolensky, "Rule induction through integrated symbolic and sub-symbolic processing", in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. 1992, pp. 969–976, Morgan Kaufmann.
 - [4] Volkmar Uebele, Shigeo Abe, and Ming-Shong Lan, "A neural-network-based fuzzy classifier", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 2, Feb. 1995.
 - [5] S. Salzberg, "A nearest hyperrectangle learning method", in *Machine Learning*, 1991, 6, pp. 251–276.
 - [6] D. Wettschereck, "A hybrid nearest-neighbour and nearest-hyperrectangle learning algorithm", in *Proceedings of the European Conference on Machine Learning*, 1994, pp. 323–335.
 - [7] J. Ross Quinlan, "Induction of decision trees", in *Machine Learning*, 1986, 1, pp. 81–106.
 - [8] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.
 - [9] P. Clark and T. Niblett, "The CN2 induction algorithm", in *Machine Learning*, 1989, 3, pp. 261–283.
 - [10] Tomohiro Takagi and Michio Sugeno, "Fuzzy identification of systems and its application to modeling and control", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 1, Jan. 1985.
 - [11] Li-Xin Wang and Jerry M. Mendel, "Generating rules by learning from examples", in *International Symposium on Intelligent Control*. IEEE, 1991, pp. 263–268.
 - [12] Charles M. Higgins and Rodney M. Goodman, "Learning fuzzy rule-based neural networks for control", in *Advances in Neural Information Processing Systems*, California, 1993, 5, pp. 350–357, Morgan Kaufmann.
 - [13] Detlef Nauck and Rudolf Kruse, "Nefclass - a neuro-fuzzy approach for the classification of data", in *Proceedings of the Symposium on Applied Computing*, 1995.
 - [14] Miguel Delgado and Antonio Gonzalez, "An inductive learning procedure to identify fuzzy systems", *Fuzzy Sets and Systems*, vol. 55, pp. 121–132, 1993.
 - [15] S. K. Halamuge and M. Glesner, "FuNe Deluxe: A group of fuzzy-neural methods for complex data analysis problems", in *Proceedings of the EUFIT'95*, Aug. 1995.
 - [16] Lotfi A. Zadeh, "Soft computing and fuzzy logic", *IEEE Software*, pp. 48–56, Nov. 1994.
 - [17] Patrick K. Simpson, "Fuzzy min-max neural networks – part 1: Classification", *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 776–786, Sept. 1992.
 - [18] Patrick K. Simpson, "Fuzzy min-max neural networks – part 2: Clustering", *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 1, pp. 32–45, Jan. 1993.
 - [19] Michael R. Berthold and Jay Diamond, "Constructive training of probabilistic neural networks", *Neurocomputing*, 1998.
 - [20] Lotfi A. Zadeh, "Fuzzy logic and the calculi of fuzzy rules and fuzzy graphs: A precis", *Multi. Val. Logic*, vol. 1, pp. 1–38, 1996.
 - [21] Klaus-Peter Huber and Michael R. Berthold, "Building precise classifiers with automatic rule extraction", in *IEEE International Conference on Neural Networks*, 1995, 3, pp. 1263–1268.
 - [22] Robert W. Blanning, "The construction and implementation of metamodels", *Simulation*, vol. 24, pp. 177–184, 1975.
 - [23] J.P.C. Kleijnen, "Model behaviour: Regression metamodel summarization", *Encyclopedia of Systems and Control*, vol. 5, pp. 3024–3030, 1987.
 - [24] Henri Pierreval, "Rule-based simulation metamodels", *European Journal of Operational Research*, vol. 61, pp. 6–17, 1992.