Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

FUZZY GRAPH BASED METAMODELING

Klaus–Peter Huber
Michael R. Berthold

IRF (Prof. D. Schmid), University of Karlsruhe
P.O. Box 6980, 76128 Karlsruhe, Germany

Helena Szczerbicka

Dept. of Computer Science, University of Bremen
P.O. Box 330440, 28334 Bremen, Germany

ABSTRACT

Analysis of simulation models has gained considerable interest in the past. However, their complexity still remains a considerable drawback in practical applications. One promising concept is the building of auxiliary models (*metamodels*) for different analysis goals. We present an efficient algorithm that constructs a metamodel only from simulation data, so no a priori knowledge has to be included. It will be shown that the resulting system approximates real valued functions with an adjustable precision. In addition the data can contain fuzzy patterns or values with a corresponding confidence–interval. This is especially well suited for simulation data due to its stochastic character. The metamodel is represented in form of a Fuzzy Graph which allows the analyst to directly extract easy to interpret if–then–rules. Application of this method to a real world token bus model is shown in detail.

1 INTRODUCTION

The use of modeling and simulation techniques has gained considerable influence for the development or the optimization of different systems. Unfortunately the complexity of the resulting simulation systems increases with the complexity of the real systems. This leads to several drawbacks: simulation becomes a highly time consuming task which makes it impossible to perform interactive simulations and in addition the analysis of the resulting simulation model is extremely complicated. Therefore there is a need for methods that help to analyze the behavior of complex models, e.g., to improve the validation process. In general two directions exist to solve this problem.

- *modelbased*: This means that the model can be analyzed based on knowledge about the concrete model structure. For example in petri net models the construction of a reachability graph helps to find places in the petri net that will never be reached.
- *databased*: If the model is executable, it is possible to perform experiments with the model e.g., by stochastic simulation of a queuing network model. The result is a huge database describing values of model parameters (factors) and corresponding values of the output parameter of interest. Analysis of dependencies between factors and the output parameters helps to build a simpler model or to better understand the behavior.

Due to the high complexity of models in real applications often there does not exist an analytical solution, so databased methods have to be used. One way is to analyze the data with statistical means like correlation analysis to find some unknown dependencies. Another way is to construct an auxiliary model, which is simpler and easier to handle than the original model. This auxiliary model is built using the example data generated by experimenting with the model and it should have the same behavior as the model.

In Blanning (1975) computation of partial derivatives was used to obtain the so–called *metamodel* from the simulation data. Depending on the task of analysis there may exist different kind of metamodels (see Figure 1), for example to analyze the parameter sensitivity (Blanning 1975; Huber and Szczerbicka 1994), to find bottlenecks (Kleijnen and Standridge 1988), or to optimize models (Yerramareddy et al. 1992). In all cases the extraction of a metamodel helps to reduce the complexity of the model that is being analyzed. Building a metamodel can be done including a priori knowledge or without it. If some underlying dependencies of parameters are known a priori, the metamodeling process can take these into account and generate a model where this information is incorporated. If this is not the case, the metamodel may be constructed only from the data set.

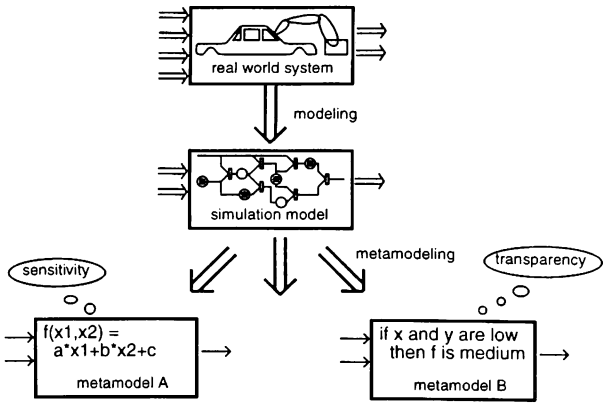

Figure 1: The Metamodeling Process

In this paper we present a metamodeling approach that only uses the data and constructs a so–called Fuzzy Graph. This allows an easy interpretation of the input–output behavior since the metamodel can be represented with fuzzy rules. Analysis of the rules helps to uncover the underlying dependencies between factors and one output parameter and the rule base can also be used as a fast simulator.

2 METAMODELING AND FUNCTION APPROXIMATION

Here, we concentrate on models where the behavior can be described by a function

$$f_{model}(\vec{x}) = f_{model}(x_1, \cdots, x_n) = y \qquad (1)$$

if n factors $x_1, \cdots, x_n$ are considered and y is the output parameter of interest. In practice this function can not be extracted explicitly from the model description and is therefore unknown. To build a suitable metamodel means to perform m experiments with the model to get m data points $(x_1^i, \cdots, x_n^i, y^i)$ $(1 \leq i \leq m)$ and to build a metamodel representing a function $f_{meta}(x_1, \cdots, x_n) = \tilde{y}$ with $\tilde{y}$ being the approximated value of y. If the data is generated by a stochastic simulation process y is a stochastic variable. There may exist a few different values of y for the same input parameters, e.g., by simulation with different random number streams. This should be taken into account when metamodeling. To validate the quality of the metamodel the difference between the observed y values and the approximated values $\tilde{y}$ can be evaluated.

Several approaches to build function approximators for metamodeling are known. They can be categorized into *statistical approaches* that use the data to adapt a special kind of function with statistical means and *Machine Learning Algorithms* that use the data to train a network or to generate a rule set.

2.1 Statistical Approaches for Metamodeling

Ad hoc methods use handfitted curves or graphical approaches where the response of the output to a changing input is represented graphically (Blanning 1975). Other methods use piecewise linear approximations (Meisel and Collins 1973) or linear regression models with a least square approach to define the parameters (Kleijnen 1979). Regression functions are very popular since the resulting models are easy to handle and interpret, and statistical methods like t-tests and F-tests can be used to validate the quality of the model (Friedman and Pressman 1988). For dealing with nonlinear behavior metamodeling regression functions are often defined as:

$$f_{meta}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \beta_j \cdot x_j + \sum_{i,l} \beta_{i,l} \cdot x_i \cdot x_l + \epsilon \quad (2)$$

with ϵ representing an error term and $\beta_{i,l}$ representing some user–defined in pair dependencies. To find a "best fitting" metamodel the regression parameters β_0, β_j and $\beta_{i,l}$ can be obtained by minimizing the mean square error due to the example data. The resulting function can be used for approximation or for analysis e.g., to obtain some information about the sensitivity of each factor.

In practice often the behavior of a simulation model can not be described with such first–order models. For example in Friedman and Pressman (1988) it is shown that the dependence between the factor *mean average numbers of customers* and the distribution parameters in a simple M/M/1–model can not be adequately described with that kind of functions. To solve this problem deep background knowledge about the model behaviour must be used. So for analysis of complex simulation models new approaches are required that build a metamodel without such a priori knowledge.

2.2 Machine Learning Approaches for Metamodeling

To avoid the need for background knowledge nowadays several methodologies have been proposed that make use of algorithms from the machine learning area. Prestructured Neural Networks are trained (Hurrion 1992; Pierreval 1996) or rule learning algorithms (Pierreval 1992) are used to build a metamodel using data from the simulation model. Although approximation with neural networks is done in a satisfying way this approach makes it hard to extract knowledge from the metamodel. Other approaches make use of rule learning algorithms to avoid this problem. In Pierreval (1992) preprocessed (i.e., cleaned and digitized) data from the simulation model was

used to directly extract decision rules and in Huber and Szczerbicka (1994) a similar approach was used for sensitivity analysis. Both concepts are based on methods that learn a classifier but not a function approximator. Therefore they are restricted to applications where the output parameter is not continuous.

Another kind of learning techniques that are especially well suited to deal with "noisy" or "stochastic" data originate from the Soft Computing or Fuzzy Systems area (Zadeh 1994). They offer an easy way to model soft data points, for example values with a corresponding confidence interval. Unfortunately, most known methods that learn fuzzy systems from data have severe limitations in this context. Some require an a priori defined set of rules that is just fine tuned during training (Uebele, Abe, and Lan 1995), others construct the ruleset during training but the result depends heavily on the order of training examples (Simpson 1993). In Wang and Mendel (1991) a simple algorithm has been proposed that divides the feature space into partitions using a grid and assigns one rule to each tile. More sophisticated algorithms (Higgins and Goodman 1993) try to divide individual attributes step by step using an increasing number of membership functions. These methods tend to be very restrictive because the feature space is split into too many tiles. Each tile represents one rule but for most of these rules no evidence was encountered during training. For the purpose of metamodeling it is of much more interest, however, to find only a few rules that cover a large portion of the feature space.

In this paper a new approach is proposed that allows the usage of *Fuzzy Graphs* (Zadeh 1994) to represent the discovered knowledge. The main advantages are the automatic and fast construction of the Fuzzy Graph based on data examples and a straightforward knowledge extraction. Since the constructed Fuzzy Graph is represented with if–then–rules the metamodel is easy to understand. Additionally, the model behavior can be approximated. This makes it possible to perform fast simulation experiments with the metamodel. The following section describes how these Fuzzy Graphs are generated and how they can be used for function approximation. An example in section 4 demonstrates how the proposed algorithm models an artificial function.

3 FUZZY GRAPHS

The central idea of a Fuzzy Graph is to represent a function with means of Fuzzy Logic instead of mathematical equations. The domains of the input parameters and the domain of the output parameter are described with so–called linguistic variables represented by individual membership functions. The mapping from input to output is defined by a set of Fuzzy Points. Well–known techniques like the center–of–gravity calculation allow one to approximate real valued functions based on this Fuzzy Graph. In this paper, a Fuzzy Graph consists of a collection of Fuzzy Points that can be represented as if–then–rules. Other ways to represent Fuzzy Graphs are described for example in Zadeh (1994). The main advantage of the Fuzzy Graph concept is the very compact and easy to understand representation of a function.

The algorithm presented in this section automatically constructs a Fuzzy Graph based on a set of training examples. The training examples can feature "soft" values y, which enables the usage of targets with confidence intervals. Fuzzy Graphs built by this approach use a fixed granularization on the dependent variable which means that the user defines the membership functions of the output variable y before training. This is helpful to focus automatic generation of the Fuzzy Graph on specific regions of the output value and to weaken constraints (and therefore the evolving number of rules) on regions with a low focus of attention. The partitioning of the input variables is determined from the training examples. The algorithm we use is derived from a constructive Neural Network training algorithm that builds a specific type of networks with locally active hidden units (Berthold and Diamond 1995). In this paper, a modified version of that algorithm is used to automatically find a set of Fuzzy Points that describe the training data (Huber and Berthold 1995).

Output granularization has to be defined before training starts. Figure 2 shows an example of a one–dimensional Fuzzy Graph constructed by the proposed algorithm. The output granularization is known

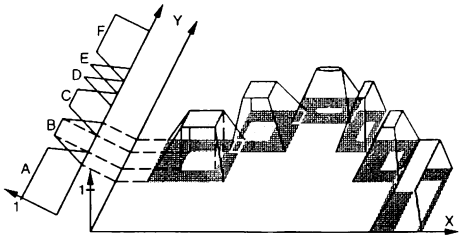

Figure 2: A One–dimensional Fuzzy Graph

a–priori: in this case 6 regions with individual membership functions are used. The used algorithm then places fuzzy points in the input space (X) to approximate the training examples. Before training starts, the output variable Y was 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.

The constructed Fuzzy Graph consists of a collection of Fuzzy Points where the *core*–region represents the smallest area where patterns of the class were found, and the larger *support*–area contains no patterns of conflict. Therefore patterns in the core–region are given a membership value of 1 while in the support–region the membership value declines linearly to 0 along its boundaries. This leads to the membership functions illustrated in Figure 3. Each

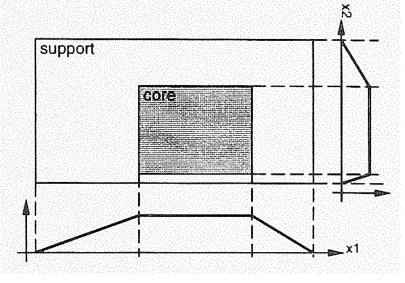

Figure 3: A Two–dimensional Fuzzy Point

such Fuzzy Point represents exactly one if–then–rule that can be described as:

IF	x_1	$\in [b_1, c_1] \subset (a_1, d_1)$
AND	$\cdots$	
AND	x_n	$\in [b_n, c_n] \subset (a_n, d_n)$
THEN		y is of class k
(weight: w)		

The input parameters are restricted through core $[b_i, c_i]$ and support–regions (a_i, d_i) with $a_i < b_i \leq c_i < d_i$ and each rule corresponds to one class k of the output parameter. Additionally, the rule weight shows the number of training examples that are covered by the core of the rule and is therefore an indication for the reliability of the rule.

3.1 Automatic Fuzzy Graph Construction

Construction of the Fuzzy Graph (or training) 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 stochastic simulation experiments. If there exist sharp targets a singleton can be used as the corresponding output membership function. From this soft target membership function μ_T the values μ_k for each class k $(1 \leq k \leq c)$ are computed, using pre–defined membership functions for each class of y. The membership values for all classes are computed using a fuzzy and–operator (min) between the target and the class membership set. Training is then performed using these target membership values. This leads to training patterns consisting of an input vector $\vec{x} = (x_1, \cdots, x_n)$ with the corresponding target $\vec{\mu} = (\mu_1, \cdots, \mu_c)$ $(0 \leq \mu_i \leq 1)$, where c denotes the number of classes.

The method makes sure that each training pattern is covered by a rule (Fuzzy Point) of the class with the highest membership value and that rules of classes with membership values = 0 do not cover the pattern. This is useful to tolerate moderately noisy patterns or small oscillations along class boundaries, as will be demonstrated later.

The training algorithm we use is based on three steps that introduce new rules when necessary and adjust the core– and support–regions of existing rules. The whole training process usually takes only about 4–5 training cycles until the structure of the Fuzzy Graph automatically stops to change. Two conditions will hold for all training patterns $(\vec{x}, \vec{\mu})$ after training: There is at least one rule of class k with the highest membership value μ_k which has $\vec{x}$ inside its core. And for all classes k with $\mu_k = 0$, $\vec{x}$ lies outside of their support area. This leads to a rule base where each pattern in the training set is covered by an appropriate rule and not covered by those of conflicting classes. The complete training algorithm is described in Huber and Berthold (1995).

3.2 Fuzzy Graph Function Approximation

To use the constructed Fuzzy Graph for function approximation the output values have to be fuzzified due to the predefined membership functions for the output parameter. These output membership functions have to be defined a priori by the user and can incorporate knowledge about the systems behavior and/or areas of interest. An example is illustrated in Figure 4.

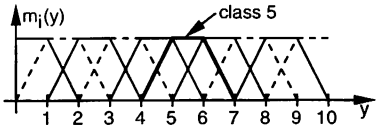

Figure 4: Fuzzification with 10 Output Classes

Based on this equidistant 10 class output–granularization 2000 randomly generated training points of an one–dimensional function were used for construction of a Fuzzy Graph. On the top of Figure 5 the Fuzzy Graph is shown and on the bottom the resulting function approximated by the Fuzzy Graph is depicted.

Especially within the context of analysing simulation data with stochastic output variables it is of interest how well a certain amount of uncertainty can

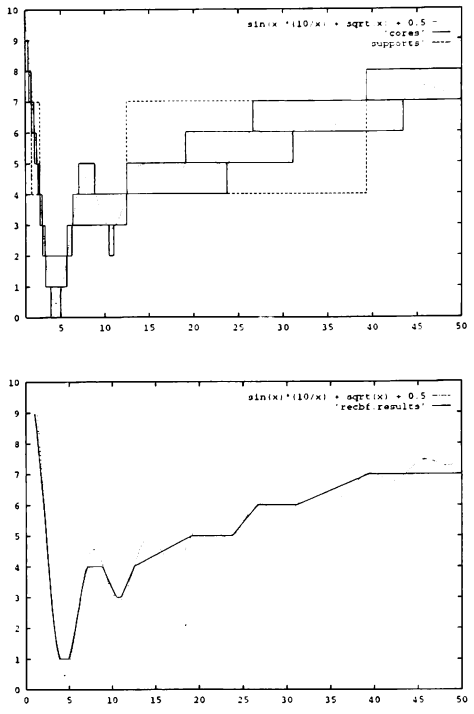

Figure 5: The Fuzzy Graph (top) and the Resulting Approximation (bottom)

be handled by this approach. A series of experiments was conducted with a specific amount of noise added to the training data to investigate the behavior of the Fuzzy Graph, for more details see Berthold and Huber (1996).

These experiments show that the resulting Fuzzy Graph approximates the original function well, with a specific degree of accuracy. In regions containing "noise", the Fuzzy Graph ignores the oscillations and tends to produce plateaus. The degree of noise tolerance depends mainly on the width of the membership functions for the output parameter. So the amount of smoothing can be controlled by the output fuzzification. Using more and finer membership functions results in higher precision but then the system tends to follow the data points very closely, and generates a lot of rules to model also the noise.

Good approximation performance helps to improve the reliability of a metamodel that is built with the presented Fuzzy Graph approach. In addition the metamodel can serve as an efficient simulation tool. Another advantage of the Fuzzy Graph metamodel is its easy to understand representation. This will be shown in the next section.

4 METAMODELING A TOKEN BUS

To demonstrate how the presented approach can be used to find rules in a real world simulation model, a token bus system was chosen. This system belongs to the class of field bus systems, i.e., a special type of communications systems, designed to connect machines and computers in a manufacturing environment. Important requirements in this area are "real time" facility, high flexibility, and low costs. In this section the analysis will mainly focus on the real time facility of the model, that is its capability to answer each request within a limited time. To guarantee this property for the given simulation model a metamodel will be built using the presented method and its behavior depending on different parameter settings will be explored.

The modeled token bus system corresponds to the seven level architecture of the ISO/OSI communication standard. Figure 6 shows the structure of the system. Many details like different message priori-

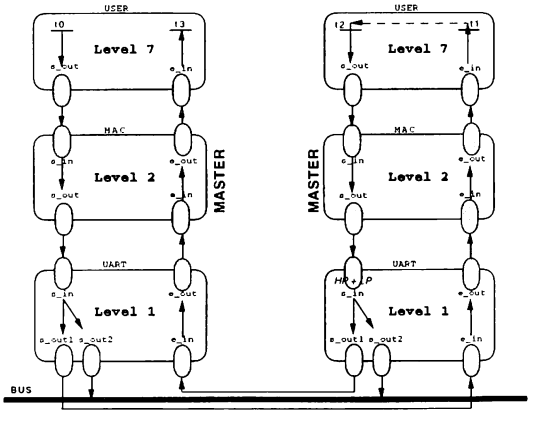

Figure 6: The Model Structure

ties 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 the UART module (Universal Asynchronous Receiver Transmitter) with its interfaces is illustrated in Figure 7. Since each station is modeled by four different modules the whole model consists of more than two hundred different queues and several hundreds connections. Due to this complexity of the internal structure a conventional analysis of this model is extremely time consuming and complicated.

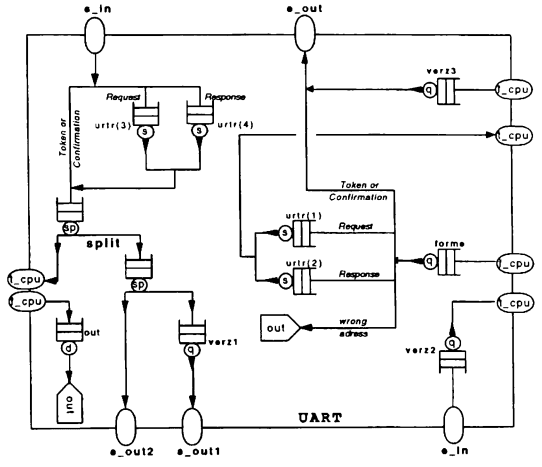

Figure 7: The Internal Structure of an UART

4.1 Metamodel Construction

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 desired that this response time always stays below an upper bound to guarantee that reaction of the system is always in time. Four 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).
- *maximum 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 training data the set of examples has to be representative. For this the planning of simulation experiments must be done carefully. In regression analysis there exist some efficient techniques for experimental design because the assumption that the model behaves linear (or in a linear way) allows to make only few experiments to determine the regression parameters (Kleijnen and Standridge 1988). With Fuzzy Graph based metamodeling no assumption about the underlying model function is made, so the selection of representative data points is not straight forward. One way is to use a randomized setting of input parameters, a more sophisticated way is to use the information of the generated fuzzy graph for iterative selection of new simulation experiments. First ideas of this new approach are described in Frank and Huber (1996). In our application a full factorial design is not possible, therefore we used randomized settings for the input parameters. In the token bus experiment we performed some simulation experiments to validate the model and to see if the model behaves stable. 352 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 (see Figure 8). With the

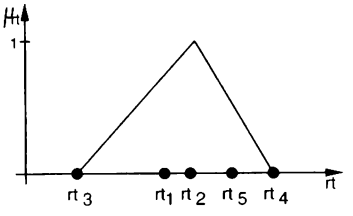

Figure 8: Generation of a Soft Target

Fuzzy Graph approach these four-dimensional data-vectors with their corresponding target membership function 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 (see Figure 9). Three series of experiments were performed with two, five and ten membership functions.

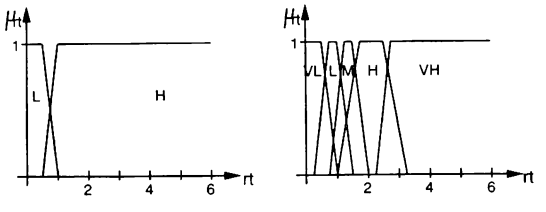

Figure 9: Two Used Types of Membership Functions

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.

4.2 Metamodeling Results

To judge the reliability of the complete rulebase the quality of the whole metamodel 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 training, the so called cross–validation set, has to be used. The dataset of 352 training vectors was split into one tenth for testing and nine tenth for training and using each tenth once for testing ten cross–validation runs were performed. The average error on the corresponding testdata was $4.4\% \pm 1.0\%$ (32 rules in average) with two, $4.1\% \pm 1.2\%$ (59 rules in average) with five, and $3.3\% \pm 0.9\%$ (68 rules in average) with ten memberships 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 classes was used for further analysis about the model behavior. In this case the classes are labeled *low* and *high*. Since the main focus of analysis are parameter settings which result in a low response time, rules of class `L = low` were investigated. From 29 rules 16 belong to this class and according to the rule weight two of the most important rules were:

IF	*cpu1*	$\in [0.11, 1.69]$	$\subset (-\infty, 1.70)$
and	*workload*	$\in [0.03, 0.99]$	$\subset (-\infty, +\infty)$
and	*trt*	$\in [0.01, 0.39]$	$\subset (-\infty, +\infty)$
and	*stations*	$\in [5, 15]$	$\subset (4, +\infty)$
THEN	class low ($rt \in [0.0, 0.5] \subset (-\infty, 1.0)$)		
(weight: 116)			

IF	*cpu1*	$\in [0.18, 2.16]$	$\subset (-\infty, 2.17)$
and	*workload*	$\in [0.03, 0.98]$	$\subset (-\infty, +\infty)$
and	*trt*	$\in [0.26, 0.39]$	$\subset (0.25, +\infty)$
and	*stations*	$\in [1, 15]$	$\subset (-\infty, +\infty)$
THEN	class low ($rt \in [0.0, 0.5] \subset (-\infty, 1.0)$)		
(weight: 24)			

Both rules demonstrate how the core always covers a confident subset of the support–region. The core of the first rule covers the whole range of two parameters, namely *workload rate* and *target–rotation–time*. 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 rate* and *performing time*. In addition the weight of this rule can be used to judge its reliability. The weight indicates the number of patterns that were covered by this rule during training. In the case of the first rule shown above 116 training patterns fall inside its core region. This means that about 36% of all training patterns 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 class. In this model the parameter *target–rotation–time* is of importance. The first rule 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. The second rule above is an indication that if the CPU is not very slow (below 2.2) a *target–rotation–time* above 0.25 also leads to low response times. In this case this parameter influences the response time.

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

IF	*cpu1*	$\in [2.71, 3.39]$	$\subset (2.70, +\infty)$
and	*workload*	$\in [0.09, 0.98]$	$\subset (-\infty, +\infty)$
and	*trt*	$\in [0.14, 0.39]$	$\subset (0.13, +\infty)$
and	*stations*	$\in [1, 15]$	$\subset (-\infty, +\infty)$
THEN	class high ($rt \in [0.5, 1.0] \subset (1.0, +\infty)$)		
(weight: 38)			

This rule indicates that if the CPU is very slow and the *target–rotation–time* 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 if high response times are to be avoided. Since only 85 training patterns are of class *high* the weight of 38 is an indication for a high reliability also of this rule.

These results illustrate the applicability of the Fuzzy Graph approach for metamodeling tasks. Since

the approximation error is acceptable the metamodel seems to be reliable and it can be used for new simulation experiments. The example rules delivered helpful information about dependencies between factors and the output of interest.

5 CONCLUSIONS

In this paper a new metamodeling approach based on Fuzzy Graphs has been proposed. With this approach it is possible to use not only real valued but also soft targets for training. This is especially well suited for the analysis of simulation models due to the stochastic character of the simulation data. It was demonstrated that the Fuzzy Graph approach is able to build suitable metamodels that can be used as a fast simulator. In addition the extracted rules deliver meaningful information about the relation between input parameters and the output, a very helpful information when analyzing simulation data of complex simulation models like the used token bus. Since the presented method is easy to handle and no parameters are needed the application of Fuzzy Graph metamodels allows to analyze complex simulation models straightforward.

REFERENCES

Berthold, M. R. and J. Diamond. 1995. Boosting the performance of RBF networks with Dynamic Decay Adjustment. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems*, 7, Cambridge, pp. 521–528. MIT Press.

Berthold, M. R. and K.-P. Huber. 1996. Automatic construction of fuzzy graphs for function approximation. In *Proceedings of the NAFIPS*, 319–323.

Blanning, R. W. 1975. The construction and implementation of metamodels. *Simulation 24*, 177–184.

Frank, E. and K.-P. Huber. 1996, September. Active learning of soft rules for system modelling. In *Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing.*

Friedman, L. W. and I. Pressman. 1988. The metamodel in simulation analysis: Can it be trusted? *Journal of the Operational Research Society 39*(10), 939–948.

Higgins, C. M. and R. M. Goodman. 1993. Learning fuzzy rule-based neural networks for control. In *Advances in Neural Information Processing Systems*, 5, California, pp. 350–357. Morgan Kaufmann.

Huber, K.-P. and M. R. Berthold. 1995. Building precise classifiers with automatic rule extraction. In *IEEE International Conference on Neural Networks*, 3, pp. 1263–1268.

Huber, K.-P. and H. Szczerbicka. 1994. Sensitivity analysis of simulation models with decision tree algorithms. In *Proceedings of the European Simulation Symposium ESS'94*, Volume 1, pp. 43–47.

Hurrion, R. D. 1992. Using a neural network to enhance the decision making quality of a visual interactive simulation model. *Journal of the Operational Research Society 43*(4), 333–341.

Kleijnen, J. 1979. Regression metamodels for generalizing simulation results. *IEEE Transactions on Systems, Man and Cybernetics 9*(2), 93–96.

Kleijnen, J. P. and C. R. Standridge. 1988. Experimental design and regression analysis in simulation: An FMS case study. *European Journal of Operational Research 33*, 257–261.

Meisel, W. S. and D. C. Collins. 1973, July. Repromodeling: An approach to efficient model utilization and interpretation. *IEEE Transactions on Systems, Man, and Cybernetics SMC-3*(4), 349–358.

Pierreval, H. 1992. Rule-based simulation metamodels. *European Journal of Operational Research 61*, 6–17.

Pierreval, H. 1996. A metamodeling approach based on neural networks. *International Journal of Computer Simulation 6*(2).

Simpson, P. K. 1993, January. Fuzzy min-max neural networks – part 2: Clustering. *IEEE Transactions on Fuzzy Systems 1*(1), 32–45.

Uebele, V., S. Abe, and M.-S. Lan. 1995, February. A neural-network-based fuzzy classifier. *IEEE Transactions on Systems, Man, and Cybernetics 25*(2).

Wang, L.-X. and J. M. Mendel. 1991. Generating rules by learning from examples. In *International Symposium on Intelligent Control*, pp. 263–268. IEEE.

Yerramareddy, S., D. K. Tcheng, S. C.-Y. Lu, and D. N. Assanis. 1992, June. Creating and using models for engineering design a machine-learning approach. *IEEE Expert*, 52–59.

Zadeh, L. A. 1994, November. Soft computing and fuzzy logic. *IEEE Software*, 48–56.

AUTHOR BIOGRAPHIES

KLAUS–PETER HUBER received his M.S. degree in Computer Science in 1988. Until 1992 he has been at Daimler–Benz AG, Stuttgart, working on expert systems, and is now a research assistant at the University of Karlsruhe.

MICHAEL R. BERTHOLD received his M.S. degree in Computer Science in 1992. In 1992 he has been a visiting researcher at Carnegie Mellon University, Pittsburgh, and joined Intel's Neural Network group in 1993. He is now a research assistant at the University of Karlsruhe.

HELENA SZCZERBICKA, Ph.D. 1982 in Computer Science from University of Warsaw, Poland. Since 1985 with the University of Karlsruhe and since 1994 professor of Computer Science at the University of Bremen, Germany. Associate member at the McLeods Institute of Simulation Sciences, co–editor of "Frontiers in Simulation" of the SCS.