Evolutionary versus Inductive Construction of Neurofuzzy Systems for Bioprocess Modelling

P. Marenbach[1] **and M. Brown**[2]

[1] Department of Control Engineering, Darmstadt University of Technology, Germany
[2] Department of Electronics and Computer Science, University of Southampton, UK

Abstract

The control and optimization of biotechnological processes is a complex task of industrial relevance, due to the growing importance attached to biotechnology. Therefore, there is an increasing use of intelligent data analysis methods for the development and optimization of bioprocess modelling and control. Since a clear understanding of the underlying physics does not exist, nonlinear learning systems, which can accurately model exemplar data sets and explain their behaviour to the designer, are an attractive approach. This paper investigates applying neurofuzzy construction algorithms to this problem and in particular compares a Genetic Programming structuring approach with a more conventional forwards inductive learning-type algorithm. It is shown that for simple problems, the inductive learning technique generally outperforms the Genetic Programming, although for large complex problems, the latter may prove beneficial.

1 Introduction

Incomplete knowledge on the dominant biological pathways as well as low availability of sensor information about the current physiological state are characteristic problems in biochemical engineering. Because mathematical physical modelling is a very difficult and time consuming task, data based modelling approaches such as artificial neural networks (ANN) have been applied to bioprocesses. Hybrid techniques in which well known, often simplified, differential equations (e.g. for cell growth) are used but unknown nonlinear kinetics that appear as parameters within these equations are taken from ANNs have proven to be a powerful approach. However, the remaining disadvantage of ANNs is the missing transparency of its "black-box" knowledge which leads to problems concerning their industrial acceptance.

In recent years, neurofuzzy data modelling algorithms have become an increasingly useful soft computing tool, as they combine the learning and structural properties of an ANN, with the rule-based explanation associated with fuzzy systems. This allows the modeller to possibly prime the network with expert knowledge in the form of fuzzy rules, train the network using the available numerical data and then validate/edit the final structure using the rule-based explanation facility. This combination of both expert knowledge and numerical data overcomes many of the criticisms which can be made about either technique individually, and should lead to higher performance and more robust solutions.

One problem with all rule-based systems is the *curse of dimensionality* which states that the resources required to evenly populate an n-dimensional space increases *exponentially* with respect to the dimension of the input space. This is exemplified by the rule base explosion which can occur in expert systems unless special structures are used to represent the knowledge and also illustrates the main problem with training and verifying conventional ANN models, where a huge amount of data may be required to produce models with a high degree of confidence over the complete input space.

A method for overcoming this problem is to use *additive* neurofuzzy systems which are composed of the sum (fuzzy OR) of several submodels. By breaking the problem up into smaller subcomponents, the:

size of the resulting model can be reduced,

generalisation abilities can be improved,

cheaper to implement,

more transparent as the fuzzy rules have less terms in their antecedents, and it may be possible to visualise the individual submodels' outputs and

redundant inputs can be easily removed.

This paper investigates two construction strategies for building these additive representations. The first is a conventional one-step-ahead optimal inductive learning algorithm which at each stage constructs a set of possible refinements and includes the best one into the model. The second algorithm uses a Genetic Programming (GP) idea which performs a stochastic search for an appropriate model structure. These two techniques are compared on a simulated time series problem and on a data set from a simulated biochemical process.

2 Neurofuzzy Construction Algorithms

Neurofuzzy systems combine the learning and structural properties of an ANN with the rule-based explanation facility of a fuzzy system. This is only possible because neurofuzzy models are restricted forms of ANNs/fuzzy systems which allow them to interpreted from either viewpoint. Neurofuzzy systems are composed of weighted, locally defined basis functions/fuzzy input sets, where the network's output is given by:

$$y = \sum_{i}^{p} a_i(\mathbf{x})\, w_i \tag{1}$$

where $a_i(\mathbf{x})$ is the i^{th} input basis function/fuzzy set and w_i is the corresponding weight which represents a local estimate of the network's output. Sometimes a linear function (instead of just a single weight) is associated with each basis function, and this is known as the Takagi-Sugeno model and has been popularised by Jang (Jang *et al.* 1997). The basis functions are generally chosen to by either Gaussian (radial basis functions) or B-splines which are locally defined, piecewise polynomial basis functions (Brown and Harris 1994).

One of the main advantages of using neurofuzzy systems is that the input basis functions performs a (nonlinear) pseudo-linearisation of the original set of input variables and training the weights is simply a *linear* optimisation problem. This means that well-known robust algorithms can be used to identify their values, and this is equivalent to training the rule confidences inside the fuzzy rule-base. It should be noted that sensible weight values *must* be identified otherwise the fuzzy rule-based interpretation will have little meaning and the system will essentially be a black-box mapping. In practice, regularisation (Bossley 1997) or some form of prior knowledge must be used to constrain the solutions found by the weight training routines.

However, conventional neurofuzzy systems still suffer from the *curse of dimensionality* (Bossley 1997), which states that the resources required to train (data) and implement (computer memory) a system increases exponentially with respect to the input space dimension. This effectively limits their application to modelling problems with no more than 4 inputs unless special structuring algorithms are used. One approach is the local linear models used in the TS and ANFIS algorithms (Jang *et al.* 1997), although this only reduces the input space dimensionality if a *gain-scheduling* approach is taken where the inputs to the local linear models are different from the inputs to the fuzzy rules. Another technique which has been used effectively for several years by statisticians is to search for *additive submodels*, effectively breaking the problem down into smaller subproblems, as illustrated in figure 1. Forming a decision by summing several subnetworks allows the designer to identify redundant inputs and global trends in each of the subnetworks, thus improving their understanding of the underlying process. In addition, all of the advantages listed in section 1 also apply. However, it should be stressed that it would be difficult for an expert to articulate this kind of structural information and it is only really feasible for a learning algorithm to try and extract this type of structure directly from the numerical training data.

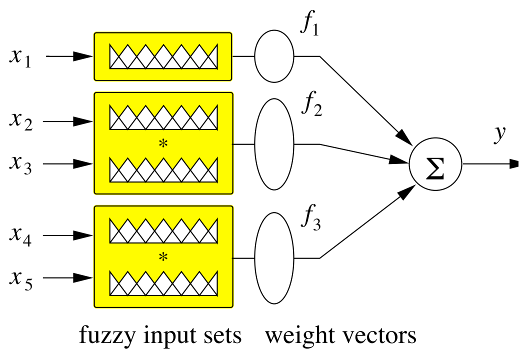

Figure 1: An additive decomposition of a network into smaller submodels.

2.1 Forwards Selection and Backwards Elimination

One technique for determining an additive structure is to use the *forwards selection* and *backwards elimination* algorithms (FSBE). These proceed in the following manner:

1. Form an initial model structure (possibly empty).
2. Evaluate whether the current model is satisfactory, if so stop.
3. Construct a set of possible model refinements.
4. Evaluate each possible refinement.
5. Incorporate the best refinement into the model, goto 2.

Where the set of refinements which increases the network's size (forwards selection) are:

add an input into the model as a separate subnetwork,

form a tensor product between two subnetworks,

add a new basis function(s) in one of the subnetworks

and the backwards elimination refinements are simply the corresponding reverse steps (basis function deletion, subnetwork split and input removing).

This procedure covers the space of possible network structures by performing a one-step-ahead optimal search. It is possible for it to stuck in local minima due to symmetries in the data, but the use of the concepts of:

store which contains different representations of each of the input variables (course, medium and fine set distribution) which can overcome most of the univariate symmetry problems and

failure margin which can overcome local minima as the algorithms can continue for a small number of iterations (failure margin) even when the performance appears to get worse, then it rewinds to the optimal network.

means that it performs well on many data sets with a small increase in the computational load. Often much of the variance in a data set can be explained using additive submodels and these inductive learning techniques generally are an efficient method to extract this information.

Typically, statistical significance measures such as Bayesian or structural risk minimisation or full leave one out cross validation are used to provide an (unbiased) measure of the network's performance or if data is plentiful, a separate independent test set may be constructed (Gunn *et al.* 1997).

3 Evolutionary Design of Fuzzy systems

In the past numerous techniques where proposed which use evolutionary algorithms for the design or optimisation of fuzzy systems. In first approaches genetic algorithms (GA) or evolution strategies (ES) where used to optimize only the numerical values, like e.g. position and shape of fuzzy sets or even rule weights, within a given fuzzy system. However, there exists a variety of very efficient conventional methods to do the same and all of these approaches do not overcome the *curse of dimensionality*. For the latter optimisation of structure is necessary which can be achieved e.g. by conventional search procedures as described above or by genetic programming (Koza 1992). GP is an extension of conventional genetic algorithms in which individuals are represented as rooted point-labeled trees of varying sizes instead of (binary) strings of fixed length used in conventional GAs. In modelling GP can be used e.g. for the generation of block oriented mathematical models (Marenbach *et al.* 1997).

Techniques which use GP to evolve fuzzy rules where described e.g. by Alba *et al.* (1996) or Steinkogler and Koch (1996). Both approaches suffer from the difficulty in appropriately representing a fuzzy rule basis as a tree structure, although *cellular encoding*, a technique proposed by Gruau (1992), can overcome this problem.

3.1 Neurofuzzy Construction using Cellular Encoding

Instead of representing the structure of a neurofuzzy system directly as a tree, cellular encoding is used to generate instructions how this structure can be developed from a very simple *embryonic* neurofuzzy network, in a manner analogous to the FSBE algorithm. GP is applied to evolve these instructions which are therefore tree structured. For each individual the embryonic network has to be modified according to its *instruction tree* before its fitness can be derived. An instruction tree consists of two types of knots

structure modifying (or refinement) knots and

input specifying knots

where preservation of syntactical constraints is guaranteed by so-called structure-preserving crossover with point typing (Koza 1994). Due to this, refinements similar to those described in section 2.1 (add input, split subnet, etc.) can be used for the evolutionary construction of neurofuzzy systems.

Within this new algorithm presented here, the root of each tree has to be a structure modifying knot (SMK). Each of those has at least one structure modifying subtree and zero or one input specifying subtree. An input specifying subtree consists only of a simple chain of input specifying knots (ISK), and this is illustrated in the following example.

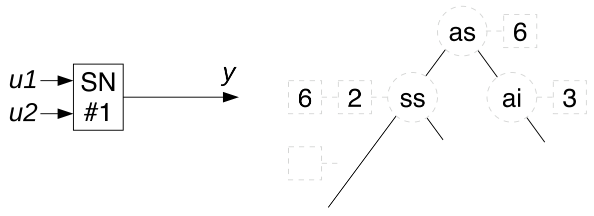

Figure 2: Embryonic neurofuzzy network (left) and instruction tree (right).

On the left in figure 2 the embryonic network with two inputs and a single output and on the right the top of an instruction tree is shown. The instruction tree consisting of SMK depicted as circles and ISK depicted as boxes is applied step by step as follows. Within the first circle “as” stands for *add subnet*. The box containing a “6” specifies that the new subnetwork shall be connected to the 6th input variable $u6$ (figure 3). Further, this refinement has two structure modifying subtrees where the left one is applied to the old (“SN#1”) and the right one is applied to the new subnet (“SN#2”).

“ss” stands for the refinement *split subnet* where the specified inputs $u2$ and $u6$ shall be taken along to the new subnetwork (“SN#3”). Because $u6$ was not connected to the original subnetwork (figure 3, “SN#1”) this part of the

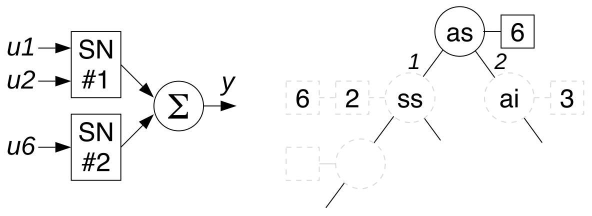

Figure 3: Neurofuzzy network after first refinement.

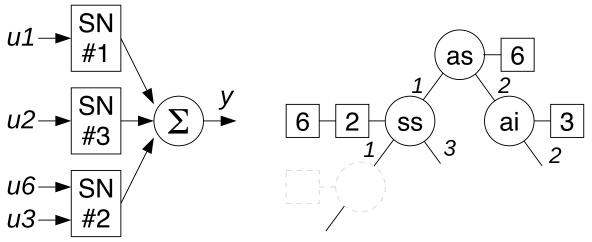

Figure 4: Neurofuzzy network after tree refinements.

instruction is ignored. The refinement "ai" = *add input* is applied to the subnetwork "SN#2": it is enlarged by input $u3$ (figure 4).

The following SMKs were implemented thus far:

add input Adds specified inputs to subnetwork

delete input Removes specified inputs from subnetwork

add subnet Adds new subnetwork with specified inputs

split subnet Splits subnetwork: the specified inputs are used for the new one

delete subnet Removes subnetwork

double knots Increases number of fuzzy sets for specified inputs

half knots Decreases number of fuzzy sets for specified inputs

The most important arguments for this kind of representation are that they provide high flexibility while also guaranteeing that similar instruction trees produce similar solutions. The latter point is especially important for evolutionary algorithms although it is often neglected in the literature. In contrast to the algorithm of Gruau (1992) in which the ANN's structure as well as it weights were generated by GP, for this application only the structure is evolved while highly efficient least squares, training techniques are used to calculate the weights.

Objectives for optimisation are the remaining mean square error of the trained network and its number of weights. As described by Marenbach *et al.* (1996), it is advantageous to evaluate the mean square error with respect to a set of data different from the training data. The inherent parallelism of evolutionary algorithms was utilized to realise a parallel implementation for UNIX computers (cf. figure 5).

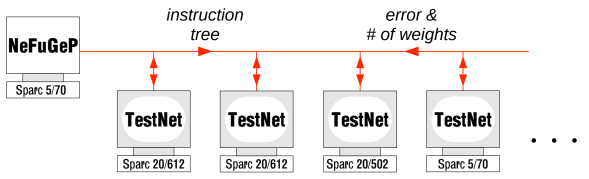

Figure 5: Parallel processing; Structure search using GP (**NeFuGeP**) and network training and evaluation (**TestNet**).

4 Examples

To give an impression of the functionality and performance of the new approach it was applied to two different modelling tasks of increasing complexity:

1. A simulated 4 input (2 delayed outputs + 2 external inputs) time series.

2. A simulated fed-batch fermentation process (10 inputs)

For sake of comparison for both approaches the Bayesian statistical significance measure calculated as

$$sSig = T\ln(MSE) + N_w \ln(T) \qquad (2)$$

was use as optimisation/fitness criterion where T is the number of training data, N_w the number of weights, and MSE the mean square error.

4.1 Simulated time series

In this example, the two neurofuzzy construction algorithms are used to model the following simulated time series:

$$\begin{aligned} y(t) \;=\; & \left(0.8 - 0.5e^{-y^2(t-1)}\right)y(t-1) - \\ & \left(0.3 + 0.9e^{-y^2(t-1)}\right)y(t-2) + u(t-1) + \qquad (3) \\ & 0.2u(t-2) + 0.1u(t-1)u(t-2) + \mathcal{N}(0, 0.01) \end{aligned}$$

where the additive noise $\mathcal{N}(0, 0.01)$ is generated from a zero-mean Gaussian distribution with variance 0.01 and $u(t)$ is the external input which is generated randomly from a uniform distribution on the interval $[-1, 1]$ (cf. figure 6). Therefore, the underlying equations can be composed into two additive functions, one involving the delayed outputs ($y(t-1)$ and $y(t-2)$) and another which is based on the external inputs ($u(t-1)$ and $(u(t-2))$. Three redundant inputs $y(t-3)$, $y(t-4)$ and $u(t-3)$ were also included in the construction algorithms as possible inputs.

The inductive learning algorithm is able to find the correct additive structure and the linear representation of the inputs $y(t-2)$, $y(t-1)$, $u(t-1)$ and $u(t-2)$, demonstrate that it has found a good model which is able to adequately predict the noisy data with a parsimonious network (21 weights, see figure 8).

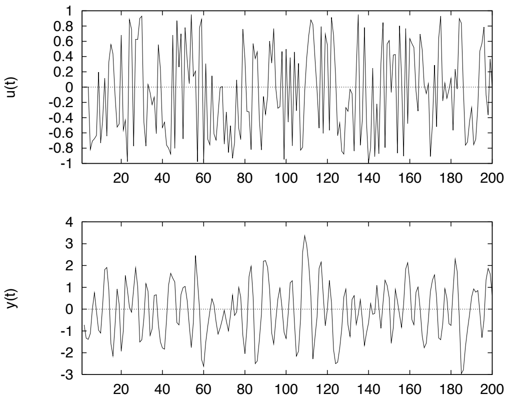

Figure 6: Input $u(t)$ and output $y(t)$ of first example.

In figure 7 the development of the statistical significance measure for several different runs is shown. After about 630 iterations of the Forwards Selection Algorithm the network depicted in figure 8 with a remaining mean square error of 0.01 (equal to the noise variance) is obtained. A number of runs of the GP approach with population size 200 and 100 individuals each produced the same basic network structure with a slightly smaller MSE but 37 weights, i.e. one fuzzy input variable got more membership functions. This can be explained by the fact that the possibilities for fine tuning of the fuzzy terms are limited within the GP approach, although as has been stated, there exists conventional algorithms which could be used for this task. The statistical significance measure of the inductive constructed model was about 1% better (cf. figure 7). However, when comparing the total number of iterations needed by both approaches it appears that the inductive approach is superior. In runs with only 50 generations with 50 individuals which would be a comparable temporal effort[1] if four parallel evaluations are assumed the optimal structure was never achieved (cf. figure 7).

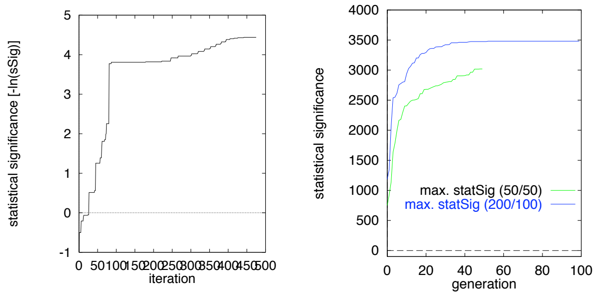

Figure 7: Statistical significance: Using the inductive (left) and the evolutionary (right) approach.

[1]Computation for this example took about 30 minutes on a SparcStation 5 using the FSBE algorithm.

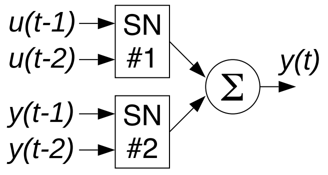

Figure 8: Optimal neurofuzzy network structure.

4.2 Simulated bioprocess

The second example process was taken from the Modelling and Control Competition initiated by the Biological Control Forum. It is a fed-batch fermentation process producing a secondary metabolite as the product which is given as a black-box simulation model. The two substrates S_1, S_2 needed by the microorganisms for growth and production are provided by two feeds.

To use learning modelling techniques together with common basic knowledge on bioprocesses, a hybrid approach was chosen, where the neurofuzzy network is applied to model the unknown nonlinear growth kinetic $\mu(t)$. Based on this value the development of the biomass $X(t)$ can be derived mathematically from:

$$\dot{X}(t) = \mu(t)X(t)\ . \qquad (4)$$

Besides S_1, S_2 and X the product concentration P, the volume V and the derivatives of each of these five variables was given as possible inputs to the network.

Again the inductive approach was able to generate a good model structure (cf. figure 9). But this time the GP approach produced a slightly better one which even improved after "local" optimisation of the fuzzy terms.

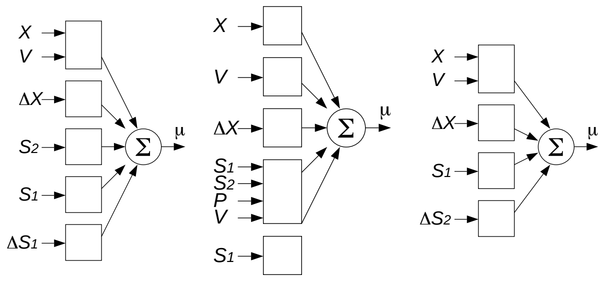

Figure 9: Optimised network structure: Constructed by inductive (left) or evolutionary approach without (middle) or with generalisation test (right).

However, this example also demonstrated that the statistical significance measure alone is not enough to prevent overfitting (see table 1). Further runs for which one data set was used for training and another one for evaluation of fitness produced a third model with provides good results even when applied to validation data that was used neither for training nor for testing (cf. figure 10). This

		training data		validation	
	N_w	MSE	$sSig$	MSE	$sSig$
FSBE	22	$7.4\,10^{-5}$	4352	$3.0\,10^{-4}$	3685
GP 1	28	$6.1\,10^{-5}$	4405	$4.7\,10^{-4}$	3447
GP 2	16	$1.0\,10^{-4}$	4242	$1.4\,10^{-4}$	4089

Table 1: Result with inductive (FSBE) or evolutionary approach without (GP 1) or with generalisation test (GP 2).

could possibly be improved further if a Bayesian approach is taken to modelling and the complete data set is used to estimate how significant each of the submodels/fuzzy rules are and use this to explicitly regularise or combine the models (Gunn *et al*. 1997).

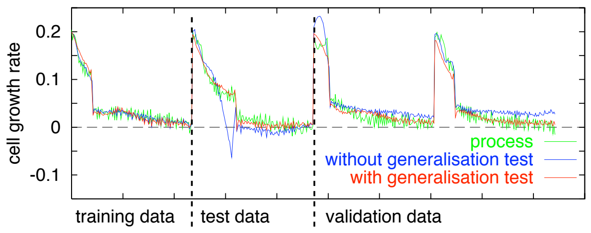

Figure 10: Process versus network output.

5 Conclusions

In this paper motivation for an application of structured neurofuzzy systems for modelling of complex nonlinear plants such as biochemical processes is given. The combination of rule-based explanation facility of a fuzzy system with the capability of learning can be helpful for the understanding and development of complex processes if and only if the complexity of an achieved rule basis is limited to the dominant and most significant process aspects. Therefore techniques for structuring of neurofuzzy systems are essential.

Both algorithms presented and compared here show promising features and in particular produced good results on both examined modelling tasks. It is shown that the inductive learning approach generally outperforms the GP in terms of speed. For highly complex processes the evolutionary approach could be advantageous although no significant differences in the achieved model quality were recognised in this work. To produce reliable models which are able to generalise, both techniques need additional information to validate the calculated solutions and to assess the performance of the refinements. This will be an important area for further work.

Finally, it is worse saying that the GP algorithms could easily be extended for the construction of deeper hierarchical network structures which could further reduce the number of necessary rules and therefore increase transparency. However, this causes training to be more difficult and time consuming training due to the loss of linearity of the network output with respect to the weights.

Acknowledgments

The authors gratefully acknowledge the financial support of the following: The German Ministry of Education and Research (BMBF), and the EPSRC GR/K55110.

References

Alba, E., Cotta, C. and Troyo, J. J. (1996). Type-constrained genetic programming for rule-base definition in fuzzy logic controllers. In: *Proc. 1st Conf. on Genetic Programming*. Stanford, CA.

Bossley, K. (1997). Neurofuzzy Modelling Approaches in System Identification. PhD thesis. Faculty of Engineering and Applied Sciences, University of Southampton. Available from: http://www.isis.ecs.soton.ac.uk/pub/theses/kmb:97/thesis.html.

Brown, M. and Harris, C. (1994). *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall. Hemel-Hempstead, UK.

Gruau, F. (1992). Cellular encoding of genetic neural networks. Research Report 92-21. Ecole Normale Supérieure de Lyon. France.

Gunn, S., Brown, M. and Bossley, K. (1997). Network performance assessment for neurofuzzy data modelling algorithms. In: *Intelligent Data Analysis*. London.

Jang, J., Sun, C. and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Prentice Hall.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press. Cambridge, Massachusetts.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press. Cambridge Massachusetts.

Marenbach, P., Bettenhausen, K. D. and Freyer, S. (1996). Signal path oriented approach to generation of dynamic process models. In: *Proc. 1st Conf. on Genetic Programming*. Stanford, CA.

Marenbach, P., Bettenhausen, K. D., Freyer, S., Nieken, U. and Rettenmaier, H. (1997). Data driven structured modelling of a biotechnological fed-batch fermentation by means of genetic programming. To appear in: *Proc. of the Institution of Mechanical Engineers: J. of Systems and Control Engineering*.

Steinkogler, A. and Koch, J. (1996). Genetic programming designs hierarchic fuzzy logic controllers. In: *Proc. Fuzzy Logic in Engineering and Natural Sciences (Fuzzy 96)*. Zittau, Germany.