

# Genetic Programming for Model Selection of TSK-Fuzzy Systems

Frank Hoffmann<sup>a</sup> Oliver Nelles<sup>b</sup>

<sup>a</sup>*Royal Institute of Technology  
Center for Autonomous Systems  
SE-10044 Stockholm  
E-mail: hoffmann@nada.kth.se*

<sup>b</sup>*SIEMENS Automotive  
AT PT DTS FDC  
Osterhofener Str. 14  
D-93055 Regensburg  
E-mail: oliver.nelles@at.siemens.de*

---

## Abstract

This paper compares a genetic programming approach with a greedy partition algorithm (LOLIMOT) for structure identification of local linear neuro-fuzzy models. The crisp linear conclusion part of a Takagi-Sugeno-Kang (TSK) fuzzy rule describes the underlying model in the local region specified in the premise. The objective of structure identification is to identify an optimal partition of the input space into Gaussian, axis-orthogonal fuzzy sets. The linear parameters in the rule consequent are then estimated by means of a local weighted least squares algorithm. LOLIMOT is an incremental tree-construction algorithm that partitions the input space by axis-orthogonal splits. In each iteration it greedily adds the new model that minimizes the classification error. Genetic programming performs a global search for the optimal partition tree and is therefore able to backtrack in case of sub-optimal intermediate split decisions. We compare the performance of both methods for function approximation of a highly non-linear two dimensional test function and an engine characteristic map.

*Key words:* Fuzzy modeling, genetic programming, neuro-fuzzy system

---

## 1 Introduction

Data analysis and knowledge discovery grow in importance as more data is gathered automatically in industrial processes, commerce, medicine and user

behavior on the internet to mention only a few areas. The accumulation of vast amounts of data increases the need for tools to obtain a model and to help a human to gain insight into the underlying process. The term data mining for this new field reflects the potential value of information that is somewhere hidden in the huge data sets.

A learned model obtained from a data set can be used for prediction, simulation, optimization, analysis and control of the underlying system [1,2]. For example, in control the identified model of a complex process is helpful to design appropriate control laws for the task at hand. An inverse model computes the control action that achieves a desired output from a given state. Often the inverse model is not well defined, in particular if the dimensionality of the output is different from the the dimensionality of possible actions. Methods such as linear quadratic regulation, nonlinear optimal control and dynamic programming utilize a forward model to predict future outputs based on the current state and control action.

Fuzzy systems are particularly intriguing for data analysis and modeling as their mode of approximate reasoning resembles the decision making employed by humans [3]. A fuzzy system is usually designed by interviewing an expert and formulating her implicit knowledge of the underlying process into a set of linguistic variables and fuzzy rules. An alternative is to extract the fuzzy rules from data. Combining the learning capability of neural networks with the knowledge representation of fuzzy logic results in neuro-fuzzy systems, which extract fuzzy rules from training data [4]. Typically, the fuzzy model is transferred into a neural network like architecture, which then is trained by some learning method, such as gradient descent. In this paper we consider Takagi-Sugeno-Kang neuro-fuzzy systems, in which the conclusion of a rule is a linear function of the inputs rather than a fuzzy set [5].

Evolutionary algorithms are optimization methods that mimic the processes that occur in natural evolution and genetics. A population of candidate solutions evolves over time by means of genetic operators such as mutation, recombination and selection. The different approaches, genetic algorithms (GA), evolution strategies (ES), evolutionary programming (EP) and genetic programming (GP), are distinguished by the genetic structures that undergo adaptation and the genetic operators by which they generate new variants. GP proposed by Koza [6] is concerned with the automatic generation of computer programs by means of simulated evolution. GP has been applied to a remarkable variety of different domains, such as symbolic regression, electronic circuit design, data mining, biochemistry, robot control, optimization, pattern recognition, planning and evolving game-playing strategies.

Evolutionary algorithms can be used to tune or to learn a fuzzy system based on design criteria such as stability, performance indices or modeling error.

This new field of genetic-fuzzy systems continues to grow in visibility [7]. Previous approaches for the evolutionary design of TSK-fuzzy systems employ a two-stage algorithm, in the generation stage an ES finds good candidate rules, in the subsequent refinement stage a GA and another ES further optimize the rules in order to improve their overall cooperation [8]. Genetic programming techniques have been successfully applied to identify the input variables, membership functions and rule base of a fuzzy model [9].

## 2 Takagi-Sugeno-Kang Fuzzy Models

A function  $\hat{y} = f(u_1, u_2, \dots, u_n)$  shall be approximated by a Takagi-Sugeno-Kang (TSK) fuzzy model [5]. The network architecture of a TSK fuzzy model is depicted in Fig. 1 where the rule premise inputs  $\underline{z} = [z_1 \ z_2 \ \dots \ z_{nz}]^T$  and the rule consequent inputs  $\underline{x} = [x_1 \ x_2 \ \dots \ x_{nx}]^T$  are subsets of the function inputs  $\underline{u} = [u_1 \ u_2 \ \dots \ u_n]^T$ . Each neuron  $i = 1, \dots, M$  of the network in Fig. 1 realizes a fuzzy rule

$$\begin{aligned} R_i : & \text{ IF } z_1 \text{ is } A_{i,1} \text{ AND } \dots \text{ AND } z_{nz} \text{ is } A_{i,nz} \\ & \text{ THEN } y = w_{i,0} + w_{i,1}x_1 + \dots + w_{i,nx}x_{nx} \end{aligned} \quad (1)$$

where  $A_{i,j}$  is the  $j$ th fuzzy set defined on the universe of discourse of input  $i$ . Each neuron or rule represents a local linear model (LLM) and an associated validity function that determines the region of validity of the LLM. The validity functions form a partition of unity, i.e., they are normalized such that

$$\sum_{i=1}^M \Phi_i(\underline{z}) = 1 \quad (2)$$

for any premise input  $\underline{z}$ . The output of a TSK fuzzy model is calculated as

$$\hat{y} = \sum_{i=1}^M (w_{i,0} + w_{i,1}x_1 + \dots + w_{i,nx}x_{nx}) \Phi_i(\underline{z}) \quad (3)$$

where the local linear models depend on  $\underline{x}$  and the validity functions depend on  $\underline{z}$ . Thus, the model output is calculated as a weighted sum of the outputs of the local linear models where the  $\Phi_i(\cdot)$  are interpreted as operating point dependent weighting factors. The model interpolates between different LLMs with the validity functions. The choice of  $\underline{x}$  and  $\underline{z}$  depends on the specific problem. In the most general case,  $\underline{x} = \underline{z} = \underline{u}$ , both contain all function inputs. In many applications, however, the variables that mainly cause the nonlinear effects are known a priori. Then, only these *scheduling variables*

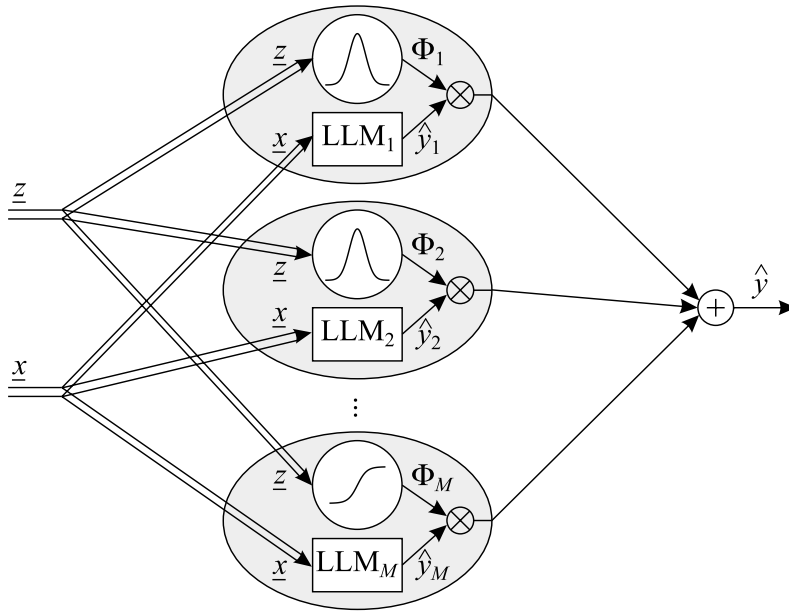


Fig. 1. Architecture of a TSK fuzzy model with  $M$  neurons for  $nx$  LLM inputs  $\underline{x}$  and  $nz$  validity function inputs  $\underline{z}$ .

have to be included in  $\underline{z}$  so that its dimension can be kept smaller than  $n$ . It is one of the major strengths of TSK fuzzy models that  $\underline{z}$  and  $\underline{x}$  can be chosen independently.

In the following, the validity functions shall be chosen as normalized Gaussians. If these Gaussians are furthermore axis-orthogonal (i.e., have diagonal covariance matrices), the validity functions become

$$\Phi_i(\underline{z}) = \frac{\mu_i(\underline{z})}{\sum_{j=1}^M \mu_j(\underline{z})} \quad (4)$$

with

$$\mu_i(\underline{z}) = \exp\left(-\frac{1}{2}\left(\frac{(z_1 - c_{i,1})^2}{\sigma_{i,1}^2} + \dots + \frac{(z_{nz} - c_{i,nz})^2}{\sigma_{i,nz}^2}\right)\right). \quad (5)$$

The rule interpretation of TSK fuzzy models in (1) depends on the choice of the centers and standard deviation in (5). Only if the premise input space partitioning is performed in an axis-orthogonal manner, the multi-variable membership functions in (5) can be projected to one-dimensional fuzzy sets  $A_{i,j}$ .

Note that the model output is *linear* in the weights  $w_{i,j}$  but is *nonlinear* in the centers  $c_{i,l}$  and standard deviations  $\sigma_{i,l}$  ( $i = 1, \dots, M$ ,  $j = 1, \dots, nx$ ,  $l = 1, \dots, nz$ ).

### 3 Estimation of the Rule Consequent Parameters

The rule consequent parameters are the weights of the local linear models. Because the TSK fuzzy model is linear in these parameters they can be efficiently estimated from data if the rule premises are given. The three most common approaches are

- *Global least squares*: This gives the best solution in terms of the squared model output error. The computational demand grows *cubicly* with the number of parameters, i.e., with the number of rules  $M$  and the consequent input space dimensionality  $nx$ . This is the most common approach which is pursued in ANFIS [4] and also in [10,11,5].
- *Local weighted least squares*: This neglects the overlap between the validity functions and estimates the parameters for each LLM separately by a least squares technique with the data weighted according to their validity. The computational demand grows only *linearly* with the number of the number of rules  $M$  and cubicly with the consequent input space dimensionality  $nx$ . Thus, local estimation is much more efficient than global estimation. The price to be paid is the introduction of a bias error. On the other hand, the variance error is decreased. This can be interpreted as a regularization effect of the local estimation which can be advantageous if the data is sparse and/or noisy or the model is very complex because it reduces over-fitting. Local estimation is applied in [12,13].
- *Product space clustering*: This is a popular approach for optimization of the rule premise structure, see Section 4. However, the Gustafson-Kessel algorithm commonly used for product space clustering automatically produces the rule consequent parameters as a by-product [14]. It can be shown that this corresponds to a total least squares solution if the clustering algorithm converges.

In this paper the local estimation approach is pursued due to its advantages in terms of speed and robustness against over-fitting. Instead of estimating all  $M(nx + 1)$  parameters simultaneously as it is done in the global approach,  $M$  separate local estimations are carried out for the  $nx + 1$  parameters of each local linear model. The parameter vector for each of these  $i = 1, \dots, M$  estimations is

$$\underline{w}_i = [w_{i,0} \ w_{i,1} \ \cdots \ w_{i,nx}]^T. \quad (6)$$

The corresponding regression matrices are ( $N$  denotes the number of data samples)

$$\underline{X}_i = \begin{bmatrix} 1 & x_1(1) & x_2(1) & \cdots & x_{nx}(1) \\ 1 & x_1(2) & x_2(2) & \cdots & x_{nx}(2) \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_1(N) & x_2(N) & \cdots & x_{nx}(N) \end{bmatrix}. \quad (7)$$

Note that the regression matrices of all local linear models  $i = 1, \dots, M$  are identical since the elements of  $\underline{X}_i$  do not depend on  $i$ . A local linear model with the output ( $\hat{y}_i = [\hat{y}_i(1) \hat{y}_i(2) \cdots \hat{y}_i(N)]^T$ )

$$\hat{y}_i = \underline{X}_i \underline{w}_i \quad (8)$$

is valid only in the region where the associated validity function  $\Phi_i(\cdot)$  is close to one. This will be the case close to the center of  $\Phi_i(\cdot)$ . Data in this region is highly relevant for the estimation of  $\underline{w}_i$ . As the validity function decreases the data becomes less relevant for the estimation of  $\underline{w}_i$  and more relevant for the estimation of the neighboring models. Consequently, it is straightforward to apply a weighted least squares optimization where the weighting factors are given by the validity function values, i.e.,

$$I_i = \sum_{j=1}^N \Phi_i(\underline{z}(j)) e^2(j) \longrightarrow \min_{\underline{w}_i} \quad (9)$$

where  $e(j) = y(j) - \hat{y}(j)$  represent the model errors for data sample. For the extreme case  $\sigma_{i,l} \rightarrow 0$ ,  $\Phi_i(\underline{z}(j))$  is either equal to 0 or 1, that is, only a subset of the data is used for the estimation of  $\underline{w}_i$ . For  $\sigma_{i,l} > 0$  all data samples are exploited for estimation but those whose validity value is close to 0 are virtually ignored.

## 4 Optimization of the Rule Premise Structure

The identification of the validity function parameters is a nonlinear optimization problem. The following strategies are commonly used for determination of these parameters.

- *Grid partitioning*: The number of input membership functions per input is either chosen by prior knowledge or determined by structure search

[11,5]. Grid partitioning severely suffers from the curse of dimensionality. To weaken its sensitivity to the input space dimensionality, the grid can be reduced to the regions where enough data is available or a multi-resolution grid can be used. All grid-based approaches are restricted to very low-dimensional problems and do not exploit the local complexity of the process to be modeled.

- *Nonlinear local optimization:* Originally, the input membership functions and the rule consequent parameters have been optimized simultaneously. The current state-of-the-art method, however, is to optimize the rule premise parameters by nonlinear local optimization and the rule consequent parameters by global least squares in a staggered approach as in ANFIS [4]. This is computationally expensive but typically yields very accurate results.
- *Product space clustering:* One of the most popular approaches applies the Gustafson-Kessel clustering algorithm to find hyper-planes in the product space, i.e., the space spanned by  $[u_1 \ u_2 \ \cdots \ u_n \ y]$ . More details on product space clustering approaches for the construction of local linear neuro-fuzzy models can be found in [14]. Although product space clustering is a widely applied method it suffers from a variety of drawbacks: (i) The computational effort grows strongly with the dimensionality of the problem. (ii) The number of clusters (i.e., rules) has to be fixed a priori. (iii) For an appropriate model interpretation in terms of fuzzy logic the multivariate fuzzy sets must be projected with accuracy losses onto univariate membership functions [14]. (Note that the more flexible input space partitioning generated by product space clustering can also be seen as an advantage of the method as far as approximation capabilities are concerned. It turns into a drawback only if a fuzzy logic interpretation with one-dimensional fuzzy sets is desired.) (iv) The rule premises and consequents must depend on the same variables, i.e.,  $\underline{z} = \underline{x}$ . This is a severe restriction that prevents many advanced concepts such as premise or consequent structure optimization or some simple ways to incorporate prior knowledge. (v) The local models are restricted to be linear.
- *Heuristic construction algorithms:* Probably the most widely applied class of algorithms increases the complexity of the local linear neuro-fuzzy model during training. They start with a coarse partitioning of the input space (typically with a single rule, i.e., a global linear model) and refine the model by increasing the resolution of the input space partitioning [10,12,13]. These approaches can be distinguished into the very flexible strategies which enable an (almost) arbitrary partitioning of the input space [12] or the slightly less flexible axis-oblique decomposition strategies [15] on the one side and the axis-orthogonal strategies which restrict the search to rectangular shapes [10,13] on the other side.

In the following, one algorithm based on a heuristic axis-orthogonal input space partitioning is discussed. Then, a new genetic programming approach for exactly the same task is proposed. The comparison between both approaches

assesses how much accuracy must be sacrificed by applying the simple heuristic.

#### 4.1 Local Linear Model Tree (LOLIMOT) Algorithm

LOLIMOT is an incremental tree-construction algorithm that partitions the input space by axis-orthogonal splits. In each iteration a new rule or local linear model (LLM) is added to the model. Thus, LOLIMOT belongs to the class of *incremental* or *growing* algorithms. It implements a heuristic search for the rule premise structure and avoids a time-consuming nonlinear optimization. In each iteration of the algorithm the validity functions which correspond to the actual partitioning of the input space are computed and the corresponding rule consequents are optimized by the local weighted least squares technique as described in Section 3. The only "fiddle" parameter which has to be specified a priori by the user is a proportionality factor  $\alpha$  for the standard deviations.

The LOLIMOT algorithm consists of an outer loop in which the rule premise structure is determined and a nested inner loop in which the rule consequent parameters are optimized by local estimation.

- (1) *Start with an initial model:* Construct the validity functions for the initial input space partitioning and estimate the LLM parameters by the local weighted least squares algorithm. Set  $M$  to the initial number of LLMs. If no input space partitioning is available a priori then set  $M = 1$  and start with a single LLM which in fact is a global linear model since its validity function covers the whole input space with  $\Phi_1(\underline{z}) = 1$ .
- (2) *Find worst LLM:* Calculate a local loss function for each of the  $i = 1, \dots, M$  LLMs. The local loss functions can be computed by weighting the squared model errors with the degree of validity of the corresponding local model according to (9) Find the worst performing LLM, that is,  $\max_i(I_i)$ , and denote  $b$  as the index of this worst LLM.
- (3) *Check all divisions:* The LLM  $b$  is considered for further refinement. The hyper-rectangle of this LLM is split into two halves with an axis-orthogonal split. Divisions in all dimension are tried. For each division  $dim = 1, \dots, nz$  the following steps are carried out.
  - (a) Construction of the multi-dimensional membership functions for both hyper-rectangles.
  - (b) Construction of all validity functions.
  - (c) Local estimation of the rule consequent parameters for both newly generated LLMs.
  - (d) Calculation of the loss function for the current overall model.
- (4) *Find best division:* The best of the  $nz$  alternatives checked in Step 3 is selected. The validity functions constructed in Step 3a and the LLMs

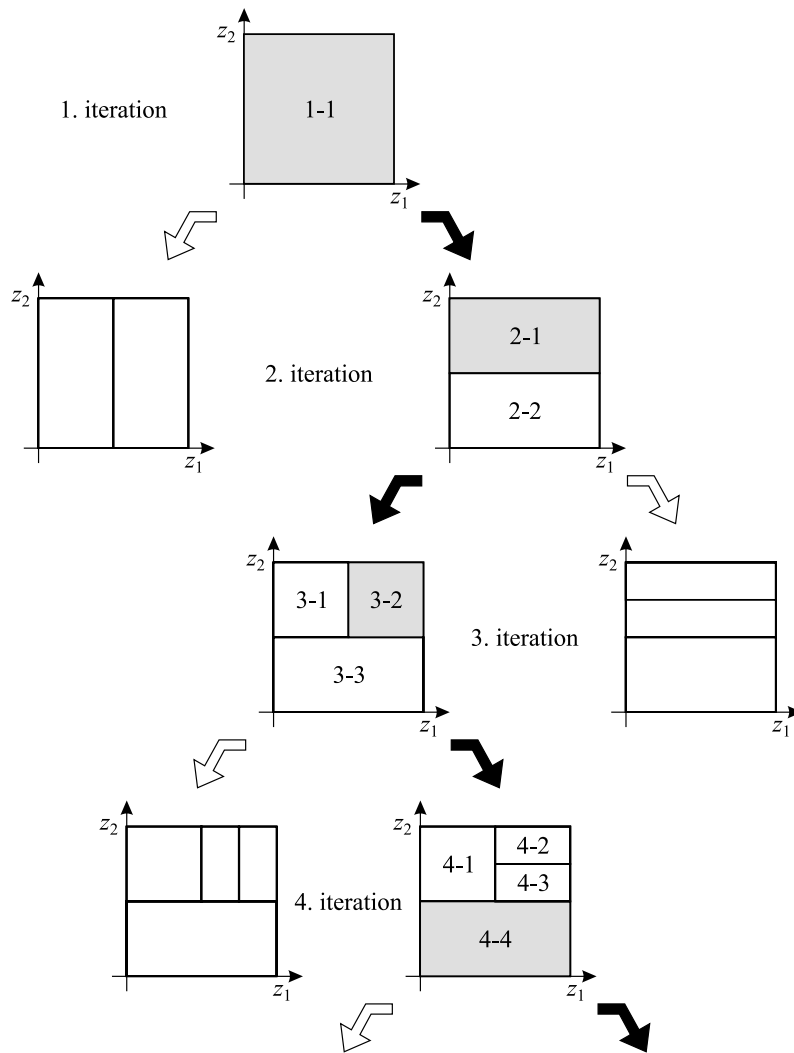


Fig. 2. Operation of the LOLIMOT structure search algorithm in the first four iterations for a two-dimensional input space ( $nz = 2$ ).

optimized in Step 3c are adopted for the model. The number of LLMs is incremented  $M \rightarrow M + 1$ .

- (5) *Test for convergence*: If the termination criterion is met then stop, else go to Step 2.

Figure 2 illustrates the operation of the LOLIMOT algorithm in the first four iterations for a two-dimensional input space and clarifies the reason for the term "tree" in the acronym LOLIMOT. Especially two features make LOLIMOT extremely fast. First, at each iteration not all possible LLMs are considered for division. Rather, Step 2 selects only the worst LLM whose division most likely yields the highest performance gain. For example, in iteration 3 in Fig. 2 only LLM 3-2 is considered for further refinement. All other LLMs are kept fixed. Second, in Step 3c the local estimation approach allows us to estimate only the parameters of those two LLMs which are newly gener-

ated by the division. For example, when in iteration 3 in Fig. 2 the LLM 3-2 is divided into LLM 4-2 and 4-3 the LLMs 3-1 and 3-3 can be directly passed to the LLMs 4-1 and 4-3 in the next iteration without any estimation.

It is important to understand that LOLIMOT is a greedy strategy or only one-step ahead optimal, that is, it maximizes in each iteration the performance gain in the next iteration. The genetic programming approach proposed next does not suffer from this sub-optimality.

#### 4.2 Local Linear Model Tree with Genetic Programming

Genetic Programming (GP) is concerned with the automatic generation of computer programs. Proposed in the early 90's by JOHN KOZA this new field has rapidly grown in visibility as a promising approach to adapt programs to particular tasks by means of artificial evolution [6]. GP improves a population of computer programs by evaluating their performance against a set of training cases. In symbolic regression for example, the quality of a tree is the mean square error between the target output and the value computed by the genetic program tree over a set of training instances. GP has been applied to a remarkable variety of different domains, such as symbolic regression, electronic circuit design, data mining, biochemistry, robot control, pattern recognition, planning and evolving game-playing strategies.

Most GP systems employ a tree structure to represent the program which computes a function or executes an algorithm. The trees in GP are composed of primitives taken from the sets of *function* and *terminal* symbols. Terminal symbols refer to nodes at the leaves of the tree and either provide a value, such as a constant or a reference to an input variable or correspond to a primitive action such as a simple robot motion command. Function symbols correspond to non-leaf nodes in the tree and compute or process an argument passed from its children nodes. Commonly used functions are arithmetic and logic operators, conditional statements and loop statements. Depending on the type of operation, unary, binary or n-ary, function nodes contain as many child nodes as arguments to the function.

[9] proposes a tree representation of fuzzy models and uses genetic programming to simultaneously identify the rule structure and the parameters of the underlying membership functions. The integration of structure and parameter identification into a single representation results in a large and complex search space. In our approach genetic programming operates in a more feasible search space as it is only used to identify the structure of the fuzzy model, whereas its optimal parameters are directly computed by means of the local least square method.

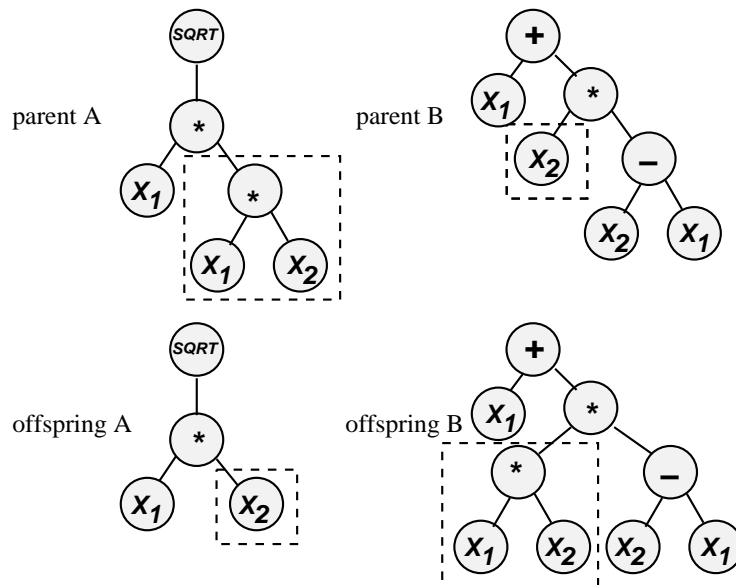


Fig. 3. Tree-based crossover.

The upper right corner of Figure 3 shows an example of a parse tree (parentB), with a terminal set  $T = \{x_1, x_2\}$  referring to input variables  $x_1, x_2$  and a function set  $F = \{+, -, *, /\}$  with the standard arithmetic operators. The tree is parsed from left to right in a depth first order resulting in the expression  $+(x_1(*(x_1(-x_2x_1))))$  in prefix notation.

The first generation of programs is seeded with a variety of tree structures. A program tree is generated by starting with the root node and iteratively assigning a symbol from the function or the terminal set to its successor nodes. The full method grows a tree by picking function symbols for the successor nodes until the current branch reaches a maximal depth and is completed with a terminal symbol at the leaf. The grow method randomly picks either a terminal or function symbol for a successor node. The resulting trees are unbalanced and contain branches of different depths.

Selection focuses the search to those regions in which good solutions already emerged. Individuals with large fitness get a better chance to reproduce offspring to the next generation. In this paper we apply tournament selection, in which the fitness of an individual is compared with a group of  $K$  other randomly selected competitors. From these candidates only the best chromosome is selected for reproduction, whereas the others are discarded. The tournament is repeated until the population is filled with a new generation of offspring.

Tree-based crossover exchanges a subtree taken from one parent with another subtree of the other parent as shown in Fig. 3. First, crossover selects a random subtree in each parent denoted by the dashed rectangle. Subsequently, the subtrees are cut off and swapped between the parents resulting in two new

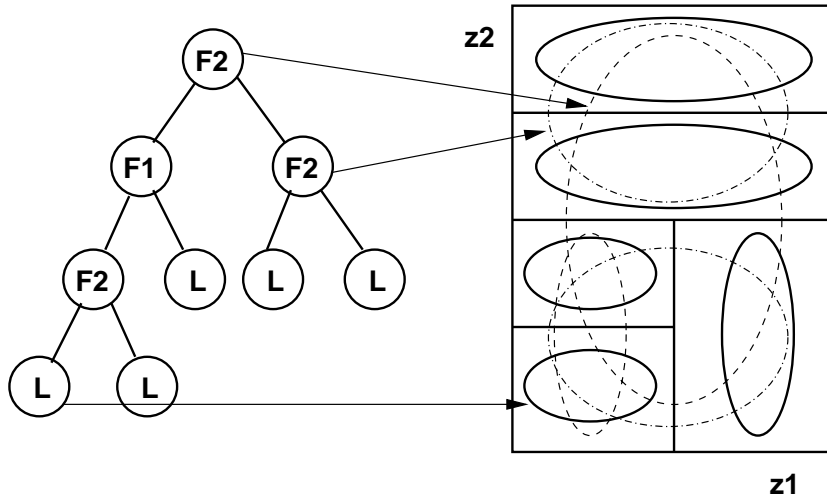


Fig. 4. GP tree and corresponding partition of input space. Solid lines denote clusters belonging to leaf nodes.

offspring. In self-crossover subtrees are exchanged within a single individual.

The mutation operator in tree-based GP replaces an existing subtree with a newly generated random subtree. Point mutation exchanges a single either function or terminal node against a random node of the same type, for example the arithmetic function  $+$  is replaced by  $*$ . Other GP mutation operators substitute terminals for subtrees, or vice versa subtrees for terminals. Our approach employs point mutation, which randomly alters the function symbol of an inner node.

The GP designer utilizes her domain specific knowledge of the problem to define the function and terminal set. Using a GP to partition the input space into clusters slightly differs from the conventional mode of computation. Instead of an algorithm or a function, each tree represents a possible distribution of cluster centers and widths. A node  $k$  from the function set  $F = \{F_i\}, i \in 1 \dots n$ , where  $n$  is the dimension of the input space, corresponds to a Gaussian fuzzy set with center  $\underline{c}^k$  and standard deviations  $\underline{\sigma}^k$ . The node function  $F_k$  denotes the dimension along which the fuzzy set is divided. The terminal set  $T = \{L\}$  only contains one generic terminal symbol to close the branch with a leaf. Figure 4 shows a possible tree and its correspondence to the resulting partition of the two-dimensional input space. Without loss of generality we restrict the discussion to the two-dimensional case for sake of better illustration. Instead of considering the standard deviations  $\underline{\sigma}_i$ , we focus our discussion on the widths  $\underline{\delta}_i$  of the hyper-rectangle which are related  $\underline{\sigma}_i = \alpha * \underline{\delta}_i$  by a single proportionality factor  $\alpha$ .

The partition scheme starts with the root node, which describes a single cluster in the center of the input space marked by a dashed ellipsoid. Similar to Step 3 of the LOLIMOT algorithm, this cluster, or hyper-rectangle, is split into two

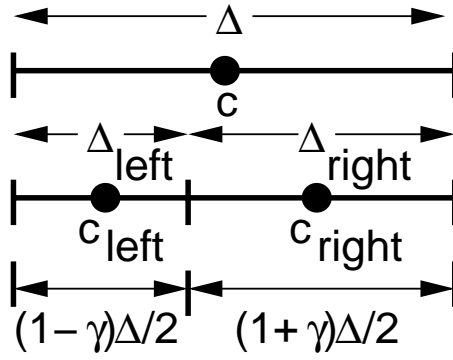


Fig. 5. Split into unequal sized halves.

hyper-rectangles along the dimension  $z_2$  specified by the function symbol  $F_2$  at the root node. Assume, we split along dimension  $k$  for a parent node  $F_k$  with center  $(c_1, \dots, c_k, \dots, c_n)$  and widths  $(\delta_1, \dots, \delta_k, \dots, \delta_n)$ . The center and width of the left and right child node then become

$$\begin{aligned}
 \underline{c_{left}} &= (c_1, \dots, c_k - \delta_k/4, \dots, c_n) \\
 \underline{\delta_{left}} &= (\delta_1, \dots, \delta_k/2, \dots, \delta_n) \\
 \underline{c_{right}} &= (c_1, \dots, c_k + \delta_k/4, \dots, c_n) \\
 \underline{\delta_{right}} &= (\delta_1, \dots, \delta_k/2, \dots, \delta_n).
 \end{aligned} \tag{10}$$

So far we assumed, that a split cuts the current cluster into equally sized halves. This constraint can be relaxed by allowing cuts that partition a hyper-rectangle into two parts of different volume. The function nodes  $F_i[\gamma]$  are augmented by an additional parameter  $\gamma \in [-1, 1]$  for the split ratio. Figure 5 illustrates the effect of an unequal split on the center locations  $c_i$  and width  $\delta_i$  of the two resulting cluster  $(\underline{c_{left}}, \underline{\delta_{left}})$  and  $(\underline{c_{right}}, \underline{\delta_{right}})$ . The cut divides into equal size halves for  $\gamma = 0$ , for  $\gamma < 0$  the *left* rectangle becomes larger, and vice versa for  $\gamma > 0$  the *right* rectangle acquires a larger volume. The new centers and widths for a split along the  $k$ -th dimension are computed as

$$\begin{aligned}
 \underline{c_{left}} &= (c_1, \dots, c_k - (1 + \gamma)\delta_k/4, \dots, c_n) \\
 \underline{\delta_{left}} &= (\delta_1, \dots, (1 - \gamma)\delta_k/2, \dots, \delta_n) \\
 \underline{c_{right}} &= (c_1, \dots, c_k + (1 - \gamma)\delta_k/4, \dots, c_n) \\
 \underline{\delta_{right}} &= (\delta_1, \dots, (1 + \gamma)\delta_k/2, \dots, \delta_n).
 \end{aligned} \tag{11}$$

Although  $\gamma$  can in principle assume any value in the continuous interval  $[-1, 1]$ , we restricted the GP search to a discrete set of possible split ratios  $\gamma \in \{-0.4, -0.2, 0.0, 0.2, 0.4\}$ .

The partition process propagates through all child nodes which are function nodes until it encounters a leaf with the unique terminal symbol  $L$ . The left-

most left node, corresponds to the small lower left cluster. The center  $\underline{c}_i$  and width  $\underline{\delta}_i$  of the leaf node are added to the list of Gaussian cluster which define the next validity function  $\Phi_l(\cdot)$ . Notice, that only leaf nodes contribute clusters (solid ellipsoids in Fig. 4), whereas inner tree nodes merely form the skeleton (dashed ellipsoids) of the input space partition. Once the GP determined the cluster centers and standard deviations, the weighted local least squares algorithm computes the optimal linear parameters in the TSK rule conclusion as described earlier in Section 3 and their normalized root mean square error (NRMSE) between the data and the model output. Obviously, the error becomes smaller as the number of clusters, and thereby the number of rules employed by the fuzzy system, increases. A neuro-fuzzy system with a large number of local models is more difficult to understand and computationally less tractable than a more comprehensive one with a fewer number of rules. This trade-off between model accuracy and complexity is taken into account by adding a penalty term for the number of local models  $\#M$  to the fitness function. The overall fitness becomes

$$F = \frac{1}{NRMSE + p \#M}. \quad (12)$$

The fitness of a fuzzy model improves if adding an additional LLM reduces the overall error by an amount larger than the penalty factor  $p$ . A small value of  $p$  favors input space partitions with a smaller approximation error at the cost of an increased number of LLMs.

## 5 Results and Comparison

In this section we compare the LOLIMOT greedy algorithm with the GP approach for approximation of a non-linear function and to model an engine characteristic map.

Figure 6 shows the training and test set error averaged over ten runs of the LOLIMOT algorithm as a function of the number of cluster centers. LOLIMOT uses 300 data points of the engine map for training and the obtained fuzzy model is evaluated on the remaining 133 data points. Although LOLIMOT does not over-fit the fuzzy model to the training data the improvement in generalization accuracy becomes marginal for more than twenty clusters. According to this observation we invoked LOLIMOT and the GP on the entire engine map data set for three different values of the penalty per model  $p = \{0.01, 0.05, 0.003\}$ . Based on similar considerations, we choose a penalty factor  $p = 0.01$  for the non-linear function in equation 13 to achieve a reasonable compromise between model accuracy and model complexity.

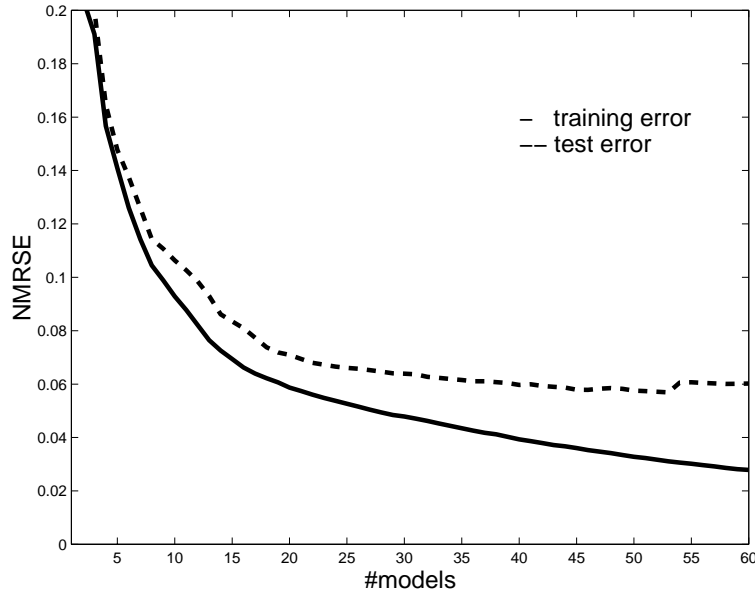


Fig. 6. Normalized root mean square error (NRMSE) over training (-) and test data set (-) in LOLIMOT for number of models  $M$ .

method	#LLM	penalty/LLM	training error	test error	fitness
LOLIMOT	18	0.01	0.2442	0.2344	2.358
GP	20	0.01	0.1936	0.1838	2.541
GP-V	18	0.01	0.2132	0.2087	2.543

Table 1

Comparison between LOLIMOT, genetic programming (GP) and genetic programming with variable split ratio (GP-V).

The GP runs for 50 generations, the population size is 400, the algorithm employs tournament selection with a tournament size of two competitors to determine the parents. Mutation changes the function symbol of a randomly selected node. Crossover swaps two subtrees among the parents.

Delgado proposed the following two-dimensional function

$$f(x_1, x_2) = 1.9 * (1.35 + \exp(x_1) * \sin(x_1 - 0.6))^2 * \exp(-x_2) * \sin(7.0 * x_2) \quad x_1, x_2 \in [0, 1]. \quad (13)$$

as a test case for TSK fuzzy system design [16].

Table 1 shows the number of models, training and test set model error and fitness for LOLIMOT, genetic programming (GP) and genetic programming with variable split ratios (GP-V). The model penalty factor of  $p = 0.01$  was chosen to limit the TSK fuzzy model complexity to a reasonable number of local linear models. As can be seen from the test set errors, the learning algorithm does

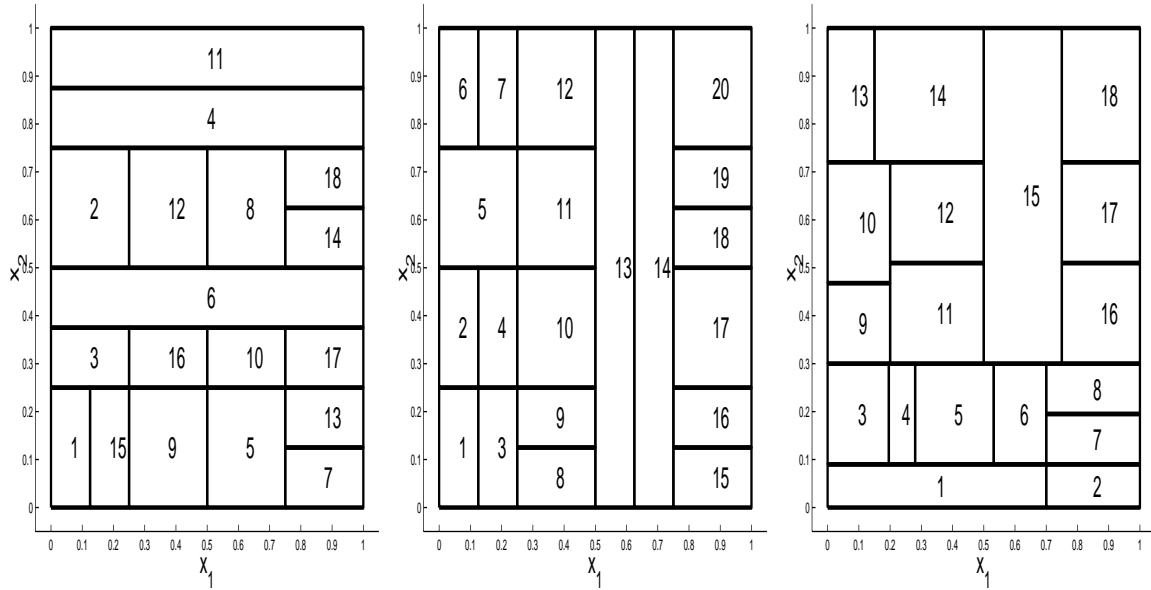


Fig. 7. Input space partition for LOLIMOT (left), GP (center) and GP with variable split ratio (right).

not over-fit the model to the training data. The best LOLIMOT partition of 18 LLMs corresponds to a fitness of  $f = 2.358$  for a NRMSE=0.2442. GP finds an improved partition with a fitness of  $f = 2.541$  for a NRMSE=0.1936 using 20 LLMs. Surprisingly, GP with variable split does not find a significantly better solution than GP with halved sized splits. It identifies an input space partition of 18 LLMs that achieves a fitness of  $f = 2.543$  for a NRMSE=0.2132.

Figure 7 shows the input space partitions generated by means of the three methods. The complexity of the search space increases with the complexity of the function to be approximated and the number of possible node symbols. This might explain why GP with a variable split ratio despite the additional degree of freedoms is not able to improve over GP with constant splits in this particular case. Running the variable GP with a larger population size over a larger number of generations might provide a remedy to this problem but on the other hand further increases the already substantial run time of the algorithm.

Figure 8 shows the map obtained by the variable split ratio GP and the modeling error between the fuzzy system approximation and the true function.

In the following we derive a TSK-fuzzy model that approximates the characteristic map of a fuel injection engine. The entire data set of 433 instances is used for training. The inputs are the engine speed in *rpm* and the injection mass in *mg*. The map output is the engine torque in *Nm*. Table 2 shows the results for LOLIMOT, genetic programming (GP) and genetic programming

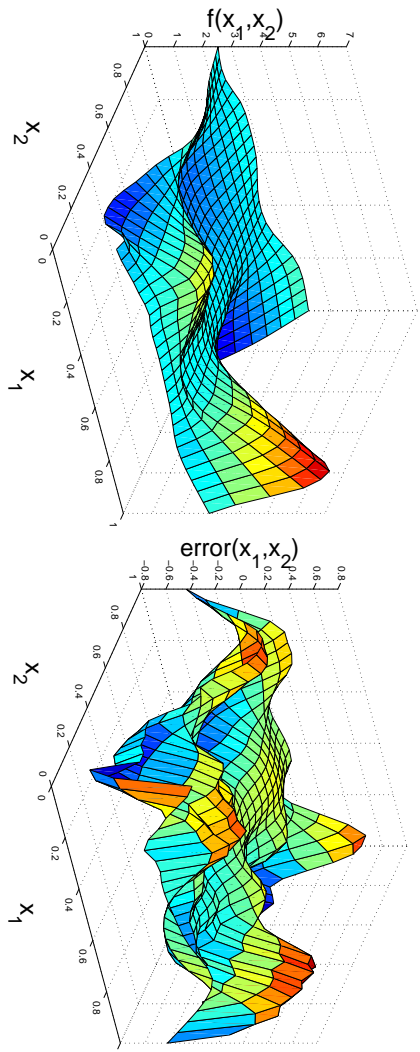


Fig. 8. Left: Function approximation using the fuzzy model obtained by GP. Right: Model error.

with variable split ratios (GP-V). The GP with a half-half split ratio found solutions that achieved a marginal better fitness than the LOLIMOT ones. The best GP tree

$$F_2 ( F_2 ( F_2 ( F_1 L L ) L ) ( F_2 L L ) ) ( F_2 L L )$$

for a large cluster penalty  $p = 0.01$  partitions the input space into 7 LLMs. The GP with variable split found clearly better solutions in terms of fitness than LOLIMOT. In the case of model penalties  $p = \{0.01, 0.003\}$  GP-V achieved a smaller error using the same or a smaller number of models than LOLIMOT. In the case of a penalty factor  $p = 0.005$  GP-V finds a better solution than LOLIMOT that although is has a larger error requires only 9 rather than 15 models. However, in this case the GP could exploit the additional parameters for the split ratio in order to improve the accuracy of the model. A fair comparison would require that LOLIMOT can choose from a set of different splits as well when it identifies the local linear model to be added next. The best partition identified with variable split ratio

$$F_2[-0.2] ( F_2[0.0] ( F_2[-0.2] ( F_1[0.0] L L ) L ) ( F_2[0.2] L L ) L )$$

reduces the model error to  $\text{NRMSE}=0.1118$  compared to LOLIMOT  $\text{NRMSE}=0.1260$  using the same number of LLMs. Three out of five nodes correspond to unequal splits along dimension  $x_2$  with split ratios  $\gamma = \pm 0.2$ .

The GP was only invoked once in the case of penalty factors  $p = \{0.005, 0.003\}$  and the results in table 2 correspond to the best individual of that single run. In the case of a penalty factor  $p = 0.01$ , both GP variants were invoked five times. The table reports the best solution found in all five runs, the values in brackets refer to the average best fitness over five runs with random seeds. In each case the GP converged to the best overall solution in two out of five runs, and found the same marginally inferior solution in the remaining three runs.

method	#LLM	penalty/LLM	error (NMRSE)	best fitness (average)
LOLIMOT	6	0.01	0.1260	5.378
	15	0.005	0.0694	6.927
	15	0.003	0.0694	8.744
GP	7	0.01	0.1155	5.390 (5.384)
	12	0.005	0.0831	6.987
	15	0.003	0.0691	8.764
GP-V	6	0.01	0.1118	5.820 (5.818)
	9	0.005	0.0886	7.513
	14	0.003	0.0685	9.054

Table 2

Comparison between LOLIMOT, genetic programming (GP) and genetic programming with variable split ratio (GP-V).

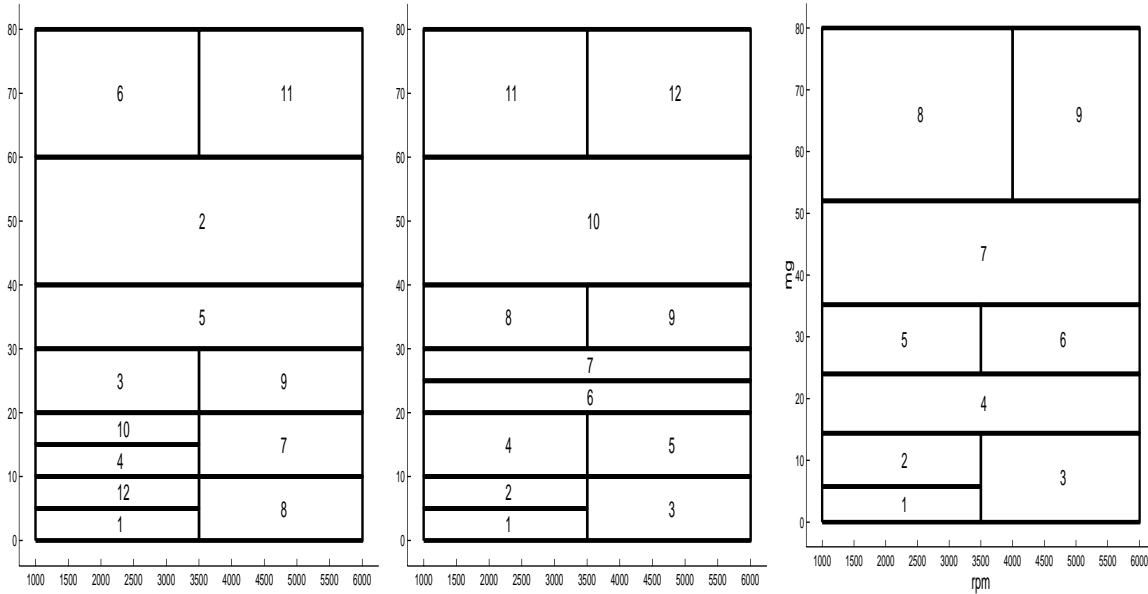


Fig. 9. Input space partition for  $p = 0.005$  for LOLIMOT (left), GP (center) and GP with variable split ratio (right). The numbers in the GP partition in the center correspond to the leaf nodes in figure 10.

For a penalty of  $p = 0.005$  GP finds a slightly better solution

$$\begin{aligned}
 &F_2 ( F_2 ( F_2 ( F_1 ( F_2 L L ) L ) ( F_1 L L ) ) ) \\
 &\quad ( F_2 ( F_2 L L ) ( F_1 L L ) ) ) \\
 &\quad ( F_2 L ( F_1 L L ) ) )
 \end{aligned}$$

with 12 LLMs, the corresponding GP tree is shown in figure 10. The numbers

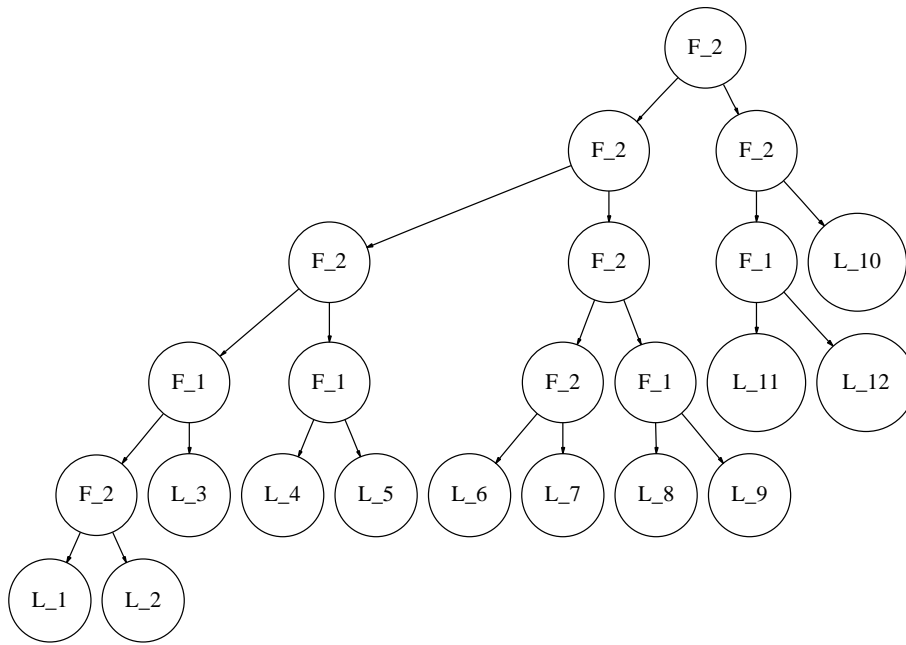


Fig. 10. Best GP tree for  $p = 0.005$ . The leaf nodes are enumerated in order to indicate the relationship to the clusters in the input partition (see figure 9).

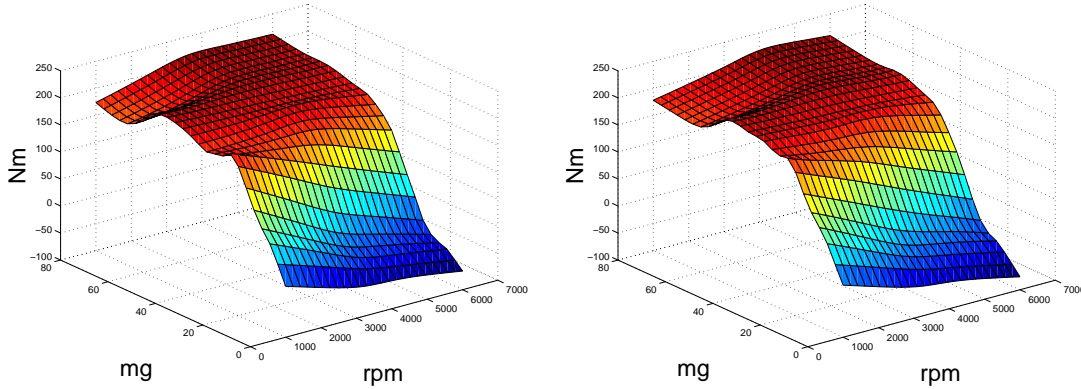


Fig. 11. Left: engine map obtained by LOLIMOT with 15 models, right: engine map obtained by variable split ratio GP with 9 models

at the leaf nodes correspond to the rectangles in the center of figure 9 depicting the resulting input space partitions for the best LOLIMOT solution (left), the best GP solution (center) and the best GP solution with variable split ratio (right). The GP tree with a variable split ratio

$$F_2[0.4] ( F_2[-0.2] ( F_1[0.0] ( F_2[0.2] L L ) L ) L ) ( F_2[0.0] ( F_2[0.2] ( F_1[0.0] L L ) L ) ( F_1[0.2] L L ) ) )$$

corresponds to an input space partition (see figure 9 right) into only 9 clusters with a model error of NRMSE=0.0881. Figure 11 shows the resulting engine map for the LOLIMOT partition with 15 LLMs and the variable split ratio GP tree with 9 LLMs.

For a small penalty term  $p = 0.003$  GP also improves the model accuracy compared to LOLIMOT. Independent of the penalty factor, the pure GP approach with halved sized splits only outperforms LOLIMOT marginally in this particular example. GP with a variable split ratio clearly demonstrates the best performance, as it utilizes the additional parameter  $\gamma$  to adjust the input space partition to the underlying data.

In case of the engine characteristic map genetic programming achieves a slight improvement in model accuracy compared with the greedy LOLIMOT method. The engine map is relatively smooth, such that it can be approximated accurately with a small number of linear models. In this case, the heuristic employed by LOLIMOT to split the cluster with the largest model error usually generates an input space partition that although not optimal is at least very similar to the best solution found by GP search.

## 6 Conclusions and Future Work

This paper introduced a genetic programming approach to structure optimization of TSK fuzzy systems. The new method is able to find improved input space partitions compared to the greedy heuristic iterative LOLIMOT scheme. The GP approach carries a larger benefit compared to LOLIMOT in case of the highly non-linear test function than for the less complex smooth engine map, which is easier to approximate.

Allowing splits into unequal size halves reduces the number of fuzzy rules while approximately achieving the same model accuracy. On the other hand GP requires a substantially larger number of model evaluations compared to the iterative LOLIMOT algorithm. In our setting, up to 20000 possible partitions are evaluated during a GP run, whereas LOLIMOT only requires about  $(D * (M - 1) * M/2)$  evaluation steps to find a good partition of a D-dimensional input space into  $M$  clusters. The additional computational burden of the GP approach is only justified in applications for which small improvements in the model accuracy or a reduction of the number of rules carry a large benefit.

The incremental construction of TSK-fuzzy models in LOLIMOT offers the advantage that it explicitly reveals the trade-off between model accuracy and the number of models. After LOLIMOT computed the error of all TSK fuzzy models up to a maximum number of rules, the user picks the model that according to his preferences offers the best compromise between model accuracy and complexity. In the GP approach the user specifies the static penalty factor  $p$  that enters the fitness function in advance, usually not knowing how many rules are required to achieve a desired accuracy. The GP framework can be extended to multi-objective function optimization, such that the population

converges towards the Pareto optimal set of solutions with respect to both criteria, namely limited number of rules and high model accuracy [17].

In the future we plan to extend the GP algorithm in a way that results in non-axis-orthogonal splits as proposed in [15]. We will also investigate genetic operators that take the local model error into account and thereby become more specific to the task of structure optimization. We will augment the GP by a multi-objective fitness function to identify the entire spectrum of Pareto optimal fuzzy models, rather than to rely on a fixed trade-off between the conflicting objectives that must be specified prior to the optimization.

## Acknowledgements

The research of the first author was supported by the NUTEK project "Architectures for Autonomous Systems" NUTEK-1k1p-99-06200 and the EU TMR Network VIRGO.

## References

- [1] C. Atkeson, A. Moore, S. Schaal, Locally weighted learning for control, *Artificial Intelligence Review* 11 (1997) 75–113.
- [2] O. Nelles, *Nonlinear System Identification*, Springer, Berlin, Germany, 2001.
- [3] M. Berthold, D. Hand, *Intelligent Data Analysis, An Introduction*, Springer Verlag, Aachen, Germany, 1999.
- [4] J.-S. Jang, ANFIS: Adaptive-network-based fuzzy inference systems, *IEEE Transactions on Systems, Man, and Cybernetics* 23 (3) (1993) 665–685.
- [5] T. Takagi, M. Sugeno, Fuzzy identification of systems and its application to modelling and control, *IEEE Trans. on Systems, Man, and Cybernetics* 15 (1) (1985) 116–132.
- [6] J. Koza, *Genetic Programming III; Darwinian Invention and Problem Solving*, Morgan Kaufman, San Francisco, CA, 1999.
- [7] O. Cordón, F. Herrera, F. Hoffmann, L. Magdalena, *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*, *Advances in Fuzzy Systems*, World Scientific, Singapore, 2001.
- [8] O. Cordón, F. Herrera, A two-stage evolutionary process for designing tsf-fuzzy rule-based systems, *IEEE Trans. on Systems, Man, and Cybernetics* 29 (6).
- [9] A. Bastian, Identifying fuzzy models utilizing genetic programming, *Fuzzy Sets and Systems* 3 (113) (2000) 333–350.

- [10] T. Johansen, Identification of non-linear system structure and parameters using regime decomposition, *Automatica* 31 (2) (1995) 321–326.
- [11] M. Sugeno, G. Kang, Structure identification of fuzzy model, *Fuzzy Sets & Systems* 28 (1) (1988) 15–33.
- [12] R. Murray-Smith, A local model network approach to nonlinear modeling, Ph.D. thesis, University of Strathclyde (1994).
- [13] O. Nelles, R. Isermann, Basis function networks for interpolation of local linear models, in: *IEEE Conf. on Decision and Control*, Kobe, Japan, 1996, pp. 470–475.
- [14] R. Babuška, H. Verbruggen, An overview of fuzzy modeling for control, *Control Engineering Practice* 4 (11) (1996) 1593–1606.
- [15] S. Ernst, Hinging hyperplane trees for approximation and identification, in: *IEEE Conference on Decision and Control (CDC)*, Tampa, USA, 1998.
- [16] M. R. Delgado, F. von Zuben, F. Gomide, Local and global estimation of takagi-sugeno consequent parameters in genetic fuzzy systems, in: *Proceedings of the North American Fuzzy Information Processing Society and the International Fuzzy Systems Association, NAFIPS/IFSA*, Vancouver, Canada, 2001.
- [17] D. Goldberg, J. Richardson, Genetic algorithms with sharing for multi-modal function optimization, in: J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, pp. 41–49.