

Structure Identification of TSK-Fuzzy Systems using Genetic Programming

Frank Hoffmann*

Royal Institute of Technology
Center for Autonomous Systems CVAP
SE-10044 Stockholm
E-mail: hoffmann@nada.kth.se

Oliver Nelles

University of California, Berkeley
Department of Mechanical Engineering
Berkeley, CA 94720
E-mail: nelles@me.berkeley.edu

Abstract

This paper explores a new approach to structure identification of Takagi-Sugeno-Kang (TSK) Fuzzy Models. We employ Genetic Programming (GP) to find an optimal partition of the input space into Gaussian, axis-orthogonal fuzzy sets. We compare the GP approach with a greedy partition algorithm (LOLIMOT) for modeling an engine characteristic map.

Keywords: Fuzzy modeling, genetic programming, data analysis

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.

Fuzzy systems are particular intriguing for intelligent data analysis, as their mode of approximate reasoning resembles the decision making employed by humans [1]. A fuzzy system is often 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[3]

Evolutionary algorithms are optimization methods that imitate the processes that occur in natural evolution and genetics. They maintain a population of candidate solutions which 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 GP proposed by Koza [5] 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 either based on some performance index or based on input-output pairs generated by the process to be modeled. This new field of genetic-fuzzy systems continues to grow in visibility. 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 [2].

⁰The author carried out the research presented in this paper at the University of California, Berkeley.

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 [9]. 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

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

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.

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

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

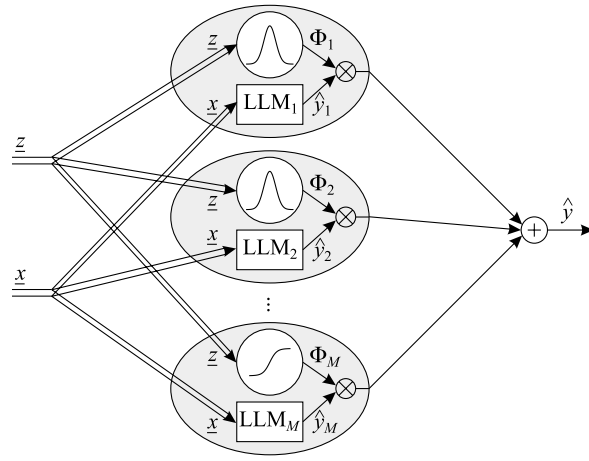


Figure 1: Architecture of a TSK fuzzy model with M neurons for nx LLM inputs \underline{x} and nz validity function inputs \underline{z} .

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 $\sigma_{i,l}$.

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 *cubically* 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 [3] and also in [4, 8, 9].
- *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 cubically with the consequent input space dimensionality nx . Thus, local estimation is much more efficient than global

estimation. Local estimation is applied in [6, 7].

- *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. 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

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

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

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

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 $(\hat{y}_i = [\hat{y}_i(1) \ \hat{y}_i(2) \ \dots \ \hat{y}_i(N)]^T)$

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

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 [8, 9]. Grid partitioning severely suffers from the curse of dimensionality.
- *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 [3]. 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 hyperplanes in the product space, i.e., the space spanned by $[u_1 \ u_2 \ \dots \ u_n \ y]$. 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. (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 [4, 6, 7].

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 allows to assess 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 α for the standard devi-

ations.

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

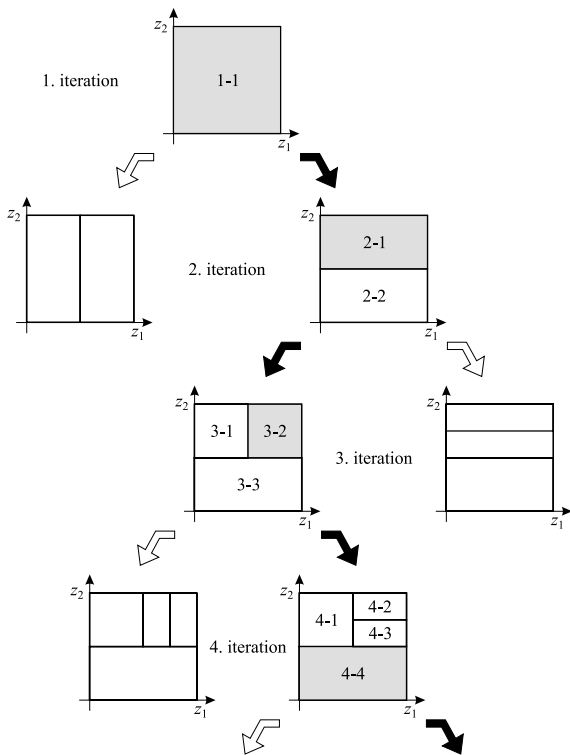


Figure 2: Operation of the LOLIMOT structure search algorithm in the first four iterations for a two-dimensional input space ($nz = 2$).

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 to estimate only the parameters of those two LLMs which are newly generated 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 [5]. GP improves a population of computer programs by evaluating their performance against a set of training cases. 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.

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.

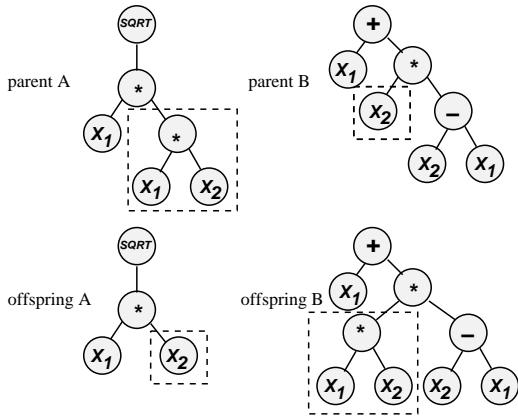


Figure 3: Tree-based crossover.

The first generation of programs is seeded with a variety of tree structures. A program tree is generated at random by selecting a symbol from the function and the terminal set for each node.

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.

Tree-based crossover swaps a subtree of one parent with a subtree of the other 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 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.

Using a GP to partition the input space into clusters slightly differs from the conventional mode of

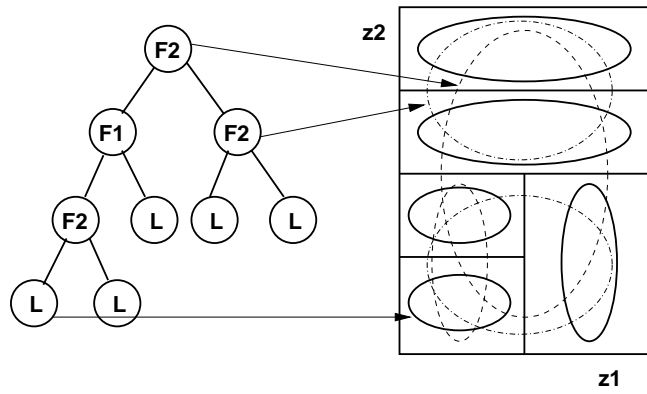


Figure 4: GP tree and corresponding partition of input space. Solid lines denote clusters belonging to leaf nodes.

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 α .

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 hyper-rectangles along the dimension z_2 specified by the function symbol F_2 at the root node.

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

tion $\Phi_l(\cdot)$. 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

$$F = \frac{1}{NRMSE + p \#M}. \quad (10)$$

The factor p defines the penalty per cluster relative to the size of the error. A small value of p results in a better approximation with the cost of an increased number of rules. The GP runs for 30 generations, the population size is 50, the algorithm employs tournament selection with a tournament size of 10 competitors to determine the parents.

5 Results and Comparison

In this section we compare the LOLIMOT greedy algorithm with the GP for modeling an engine characteristic map. The inputs are the engine speed in *rpm* and the injection mass in *mg*. The map output is the engine torque in *Nm*. The training data consists of 433 input-output pairs. We ran both algorithms for three different values of $p = \{0.01, 0.05, 0.003\}$. Table 1 shows the results for both approaches. For a medium and large penalty GP is able to find a slightly better solution than LOLIMOT. The best GP tree for a large cluster penalty $p = 0.01$,

$$(F_2 (F_2 (F_2 L L)(F_2 L L))(F_2 L L)) \quad (11)$$

method	#M	penalty/mod.	error	fitness
GP	6	0.01	0.1259	5.380
	10	0.005	0.0942	6.934
	13	0.003	0.0801	8.392
LOLI-MOT	6	0.01	0.1260	5.388
	10	0.005	0.0962	6.841
	15	0.005	0.0694	6.927
	15	0.003	0.0694	8.744

Table 1: Comparison between GP and LOLIMOT

partitions the input space along the second variable injection mass into six local models.

For $p = 0.005$ the GP found a better solution

$$(F_2 (F_2 (F_2 (F_1 L L)(F_1 L L))(F_2 L L))(F_1 L L))(F_2 L L)) \quad (12)$$

with 10 local models, compared to the best LOLIMOT solution with 15 models. The LOLIMOT solution with 10 clusters is suboptimal with a larger error than the GP model of the same size. Figure 5 shows the resulting partitions for the best GP solution (left), the best LOLIMOT solution (left) with, and the LOLIMOT solution for ten models (right). For a small penalty term $p = 0.003$ LOLIMOT clearly outperformed the GP algorithm. As the number of local models increases the search space becomes larger and GP is not able to find an optimal solution reliably. LOLIMOT offers the advantage that a previous suboptimal split can be corrected in subsequent splits and therefore does not substantially degrade the overall performance of the algorithm. Figure 6 shows the resulting engine map for the best GP solution (left) and the best LOLIMOT fuzzy system.

6 Conclusions and Future Work

In this paper we presented a new approach to structure optimization of TSK fuzzy systems. For fuzzy systems with a few number of models the GP was able to find slightly better partitions than the greedy LOLIMOT algorithm. On the other hand, GP is less robust than LOLIMOT in the sense that it does not consistently finds good solutions, especially as the number of rules increases.

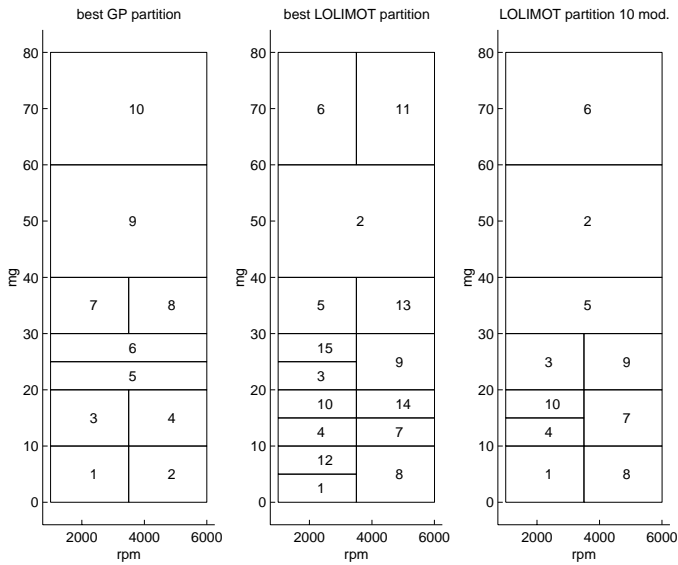


Figure 5: Left: best partition obtained by GP, center: best partition obtained by LOLIMOT, right: partition with ten clusters obtained by LOLIMOT

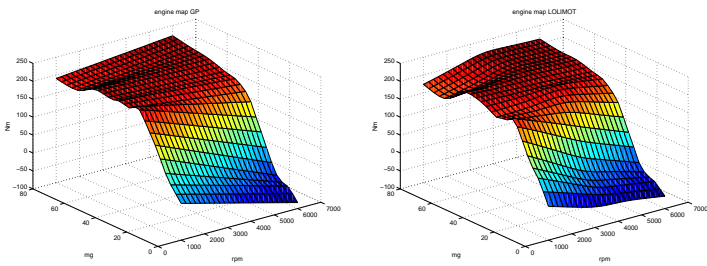


Figure 6: Left: engine map obtained by GP, right: engine map obtained by LOLIMOT for $p = 0.005$

GP requires a substantially larger amount of computation compared to the sequential LOLIMOT algorithm. This computational burden can only be justified in applications for which small improvements of accuracy carry a large benefit.

In the future we plan to extend the GP algorithm in a way that results in more general partitions. Instead of a split that cuts the current cluster into equally sized halves, one can imagine cuts that result in hyper-rectangles of unequal size. In that case, each function node will carry an additional real valued parameter that determines the ratio between the size of the left and right part. In addition one can allow non-axis-orthogonal partitions. 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.

Acknowledgements

The work of the first author was supported by the DFG (Ho 1790/2-1), by the Army Research Office (DAAH 04-96-1-0341) and the Office of Naval Research (N00014-97-1-0946).

References

- [1] M. Berthold and D. Hand. *Intelligent Data Analysis, An Introduction*. Springer Verlag, Aachen, Germany, 1999.
- [2] O. Cordón and F. Herrera. A two-stage evolutionary process for designing TSK fuzzy rule-based systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 29(6), 1999.
- [3] J.-S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, 1993.
- [4] T.A. Johansen. Identification of non-linear system structure and parameters using regime decomposition. *Automatica*, 31(2):321–326, 1995.
- [5] J.R. Koza. *Genetic Programming III; Darwinian Invention and Problem Solving*. Morgan Kaufman, San Francisco, CA, 1999.
- [6] R. Murray-Smith. *A Local Model Network Approach to Nonlinear Modeling*. PhD thesis, University of Strathclyde, 1994.
- [7] O. Nelles and R. Isermann. Basis function networks for interpolation of local linear models. In *IEEE Conf. on Decision and Control*, pages 470–475, Kobe, Japan, 1996.
- [8] M. Sugeno and G.T. Kang. Structure identification of fuzzy model. *Fuzzy Sets & Systems*, 28(1):15–33, 1988.
- [9] T. Takagi and M. Sugeno. Fuzzy identification of systems and its application to modelling and control. *IEEE Trans. on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.