Genetic Learning of Serial Hierarchical Fuzzy Systems for Large-Scale Problems

Alicia D. Benítez and Jorge Casillas[1]

[1] Department of Computer Science and Artificial Intelligence
CITIC-UGR (Research Center on Information and Communications Technology)
University of Granada, E-18071 Granada, Spain
Email: aliciades@gmail.com, casillas@decsai.ugr.es

Abstract— *When we face a problem with a high number of variables by a standard fuzzy system, the number of rules increases exponentially and then the obtained fuzzy system is scarcely interpretable. This problem can be handled by arranging the inputs in hierarchical ways. The paper presents a multi-objective Genetic Algorithm that learns Serial Hierarchical Fuzzy Systems with the aim of coping the curse of dimensionality. By means of an experimental study, we have observed that our algorithm obtains good results of interpretability and precision with problems in which the number of variables is relatively high.*

Keywords— Curse of dimensionality, hierarchical fuzzy systems, multi-objective genetic algorithms, variable selection.

1 Introduction

If a conventional fuzzy system is applied to large-scale problems (i.e, those with a high number of input variables), the number of rules grows exponentially with respect to the number of inputs received [1, 2]. Indeed, if we have n variables and k linguistic terms per variable, it requires up to n^k rules to build a complete Mamdani-based fuzzy system, and consequently, the accuracy-interpretability balance would be broken. This problem is know as the "curse of dimensionality."

In order to solve it, several approaches have been suggested such as variable selection [3, 4] and rule set reduction [5, 6]. Nevertheless, when the number of variables increases considerably, this kind of reduction is not enough to solve this problem. There exists a different approach which deals with this problem: Hierarchical Fuzzy Systems (HFS). An HFS is made up of a set of fuzzy subsystems or modules. These modules are linked in such a way that the output of a module is the input of other ones. Thanks to the decomposition of the fuzzy system made in an HFS, the complexity of each module is significantly reduced. There are several kinds of modules:

- SISO (Single Input Single Output): It has one input and one output.
- MISO (Multiple Inputs Single Output): It has several inputs and a single output [7]. We can find FLU (Fuzzy Logic Unit) in this kind of modules. One FLU special case is the one with two inputs and one output, which is equally found in the literature [2, 8–11].
- MIMO (Multiple Inputs Multiple Outputs): They have several inputs and outputs [12].

Apart from distinguishing several kinds of modules, it is also possible to find different types of hierarchical structures. There are different classifications [9, 13], though the most general one is the following [11]:

- Serial HFS (SHFS): The input of one module is the output of the previous ones, along with external variables [10].
- Parallel HFS: This system is organized in layers. The first one is made up of a set of modules receiving the input variables. Each variable is used as input only in a single module. The output of the modules in the first layer is the input of the modules which constitute the next layer, and so on. An aggregate operation might also exist in order to combine the outputs of one layer [2, 12].
- Hybrid HFS: This type of HFS is a mixture of the two previous ones [7, 14].

Other approaches study the best way to deal with the output variables in the modules (through fuzzy numbers [9] and matrixes modeling [11]) and the rules in different hierarchical levels [15]. Besides, some metaheuristics have been applied in this field with the purpose of obtaining an HFS capable of getting the best balance between precision and interpretability. For instance, Differential Evolution [10] has been employed to find the best membership functions, Genetic Algorithms [8], Ant Systems [14], Descendent Gradient Method [7] have been used to learn the hierarchical structure.

In this paper, a multi-objective genetic fuzzy system is suggested to obtain the best distribution of input variables and modules belonging to SHFS removing variables by crossover and mutation operators, so it can face the "curse of dimensionality." The paper is organized in the following sections: in Section 2, the suggested algorithm is described; in Section 3, the empirical study is shown; in Section 4, conclusions and future works are referred.

2 GSHFS algorithm

We propose a SHFS structure learning algorithm. These systems are a set of linked modules where the output of a module is one of the inputs of the next module. A module has its own fuzzy rule set which allows to infer the output variable from the input variables.

The algorithm uses a multi-objective genetic algorithm called Genetic SHFS (GSHFS) where there are two objectives to minimize: the root mean squared error (RMSE) and the number of rules. Our algorithm has three specific genetic operators: one crossover operator and two mutation operators.

ISBN: 978-989-95079-6-8

The following subsections detail the different components of the algorithm.

2.1 Coding scheme

In our algorithm, each individual represents a SHFS. The coding consists of a gene concatenation. Each gene has two fields: a variable index and a flag. The flag takes binary values (0 if it is an exogenous variable and 1 if it is an endogenous variable). Figures 1, 2, and 3 show some examples of the coding scheme.

A variable with 1 in the flag field means that is an endogenous variable (variable linking two modules) and a module exists. This variable is the input of the next module. All treated variables are variables of the problem. The algorithm decides if a variable of the problem will be endogenous or exogenous, but does not create new endogenous variables. In this way, the correlation between the variables of the problem is considered. The length of chromosome is fixed to the number of input variables of the problem. The highest hierarchy level happens when all modules are SISO. In this case, all variables will be endogenous except the first and the number of levels is equal to the number of input variables.

2.2 Initialization

The algorithm randomly generates an initial population. The variable index and the endogenous/exogenous flag is randomly chosen for each gene. The only restriction is that the variable can not be repeated and the flag of the first gene can not be 1 (i.e., the first variable have to be exogenous).

2.3 Crossover operator

The crossover operator is applied according to a probability between paired parents and provides exploitation. When crossover operator is applied to two parents, P_1 and P_2, there are a set of types of combinations which we must consider: 1) both parents, P_1 and P_2, have several modules; 2) parent P_1 has several modules and parent P_2 has one module; 3) parent P_1 has one module and parent P_2 has several modules (this case is similar to previous case); 4) both parents, P_1 and P_2, have one module. A parent-centric crossover has been used in some cases where the offspring mainly inherits the information of one of the parent and takes the secondary parent in order to add diversity.

The crossover is applied to individuals depending on the types of variables that two parents have in common. Each case has a priority. In this way, the crossover operator is applied from high to low priority as follows:

2.3.1 Both parents, P_1 and P_2, have several modules

- *Priority 1.- Endogenous-Endogenous case.* If P_1 parent has common endogenous variables with P_2 parent, a common endogenous variable is selected at random. This variable is the crossover point. The offspring O_1 is generated centered on P_1. Thus, the offspring O_1 inherits from P_2 the variables and modules from the beginning to the crossover point, but excluding the crossover point. The remaining is taken from P_1 and repeated variables are removed from the part inherited from P_2. The offspring O_2 is generated in the same way but centered on P_2.

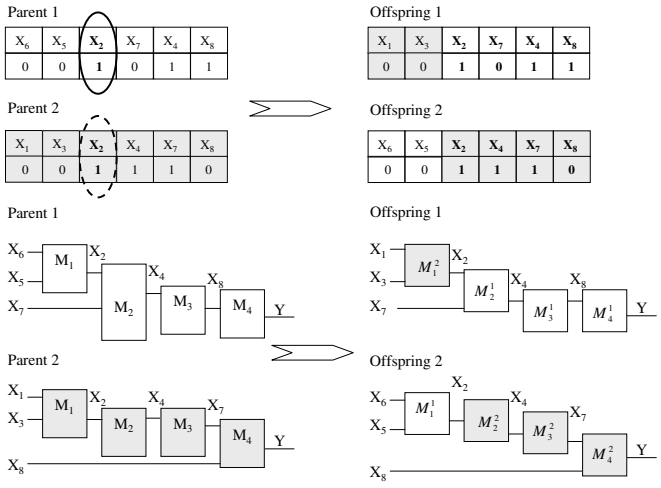

Figure 1: Example of crossover of two parents with several modules, Endogenous-Endogenous case (priority 1)

- *Priority 2.- Endogenous-Exogenous case.* If P_1 has endogenous variables which are exogenous in P_2, a common variable is randomly selected as crossover point. The offspring O_1 is generated centered on P_1. The common endogenous variable inherited from P_1 is converted into exogenous and the rest of previous modules to this variable are removed. The idea of this type of crossover is that if an endogenous variable of P_1 is equal to an exogenous variable of P_2, it indicates us that if we convert this endogenous variable into exogenous variable, RMSE will decrease, because the SHFS does not carry any error, taking the variable directly as an input.

- *Priority 3.- Exogenous-Endogenous case.* If P_1 has exogenous variables which are endogenous in P_2, one of those variables is randomly selected as crossover point. The offspring O_1 is created centered on P_1, but excluding the crossover point. The first part of O_1 is inherited from the first part of P_2 (from the first gene to the crossover point). The repeated variables are removed in O_1 from the part inherited from P_2. Figure 2 shows an example of this case. The generated SHFS offspring has four modules.

- *Priority 4.- Exogenous-Exogenous case.* If P_1 has a set of exogenous variables in common with exogenous variables in P_2, the offspring O_1 is generated as copy of P_1, but with a change: a common exogenous variable between P_1 and P_2 is randomly chosen. This variable in P_2, along with others variables, generates an endogenous variable. This endogenous variable in P_2 is selected to replace the random exogenous selected variable in O_1, which is common in P_1 and P_2. Later, the repeated variables in O_1 in part inherited from P_1 are removed. If this case comes true in P_2, O_2 offspring will be generated with the same procedure, but centered on P_2. Notice that the common exogenous variables of P_1 are looked for in P_2 excluding the exogenous variables of the module with output Y in P_2, because if this module is included, when an exogenous variable is selected there is not endogenous variable as output because it coincides with the output of

ISBN: 978-989-95079-6-8

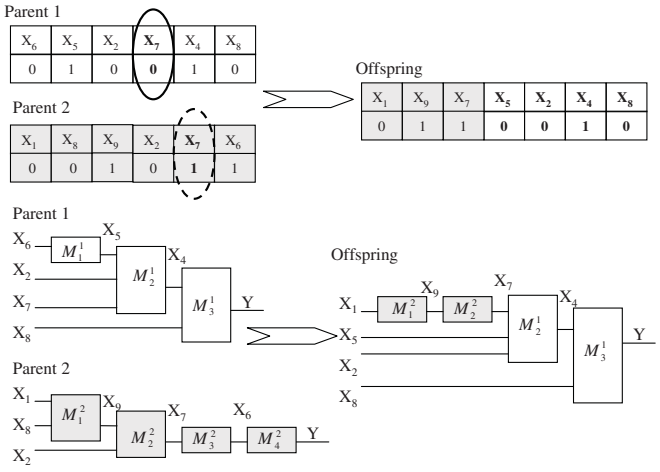

Figure 2: Example of crossover of two parents with several modules, Exogenous-Endogenous case (priority 3)

the SHFS.

- *Priority 5.- Different variables case.* If variables in P_1 and P_2 are different, an endogenous variable from P_1 and P_2 is chosen at random as crossover point. The offsprings are generated like priority 1.

2.3.2 *Parent P_1 has several modules and parent P_2 has one module*

- *Priority 1.- Endogenous-Exogenous case.* If P_1 has common endogenous variables with the exogenous variables of P_2, the offspring O_1 is generated like Endogenous-Exogenous case when P_1 and P_2 have several modules.

- *Priority 2.- Exogenous-Exogenous case.* If P_1 and P_2 have common exogenous variables, the offspring O_1 centered on P_1 is created as follows. Firstly, O_1 is generated as a copy of P_1. Then, an exogenous common variable of P_1 and P_2 is randomly chosen and moved to module with output Y. This type of crossover carries lower error and the output of SHFS is better.

- *Priority 3.- Different variables case.* In this case, O_1 is a copy of P_1 and later, an exogenous variable of P_2 is selected at random and inserted in the module of O_1 with the output Y.

2.3.3 *Parent P_1 has one module and parent P_2 has several modules*

- *Priority 1.- Exogenous-Endogenous case.* If P_1 has exogenous variables which are endogenous in P_2, the offspring O_1 is generated like Exogenous-Endogenous case when P_1 and P_2 have several modules.

- *Priority 2.- Exogenous-Exogenous case.* If P_1 has a set of exogenous variables in common with exogenous variables in P_2, the offspring O_1 is generated like Exogenous-Exogenous case when P_1 and P_2 have several modules.

- *Priority 3.- Different variables case.* If variables in P_1 and P_2 are different, an exogenous variable from P_1 and an endogenous variable from P_2 are chosen at random as crossover point. The offsprings are generated like priority 1.

2.3.4 *Both parents, P_1 and P_2, have one module*

This case, the offsprings O_1 and O_2 are generated as follows. The common exogenous variables of P_1 and P_2 are inserted in both offsprings. The rest of variables (no common variables between parents) are inserted in a set V with $|V|$ size. A number r is randomly generated: if there are not common variables between P_1 and P_2 then $r \in \{1, \ldots, n-1\}$; otherwise, $r \in \{0, \ldots, n\}$. r variables are randomly took out and inserted in O_1. Finally, the rest of variables are inserted in O_2.

2.4 *Mutation operator*

The mutation operator makes local changes in the chromosome. Two kinds of mutations have been designed:

2.4.1 *Exchange Mutation*

This operator makes an exchange of variables in a module. It chooses an exogenous variable at random in a module and exchanges between this exogenous variable and the endogenous variable of the module.

2.4.2 *Insertion Mutation*

The mutation operator distinguishes between: used and unused variables in the individual. According to it and by means of a probabilistic decision (Algorithm 1 shows a scheme), the mutation operator will choose between inserting an unused variable in a module, removing an used variable, or moving an used variable to other module.

Algorithm 1 Insertion Mutation

```
r = U[0,1];
if (r < 0.5 and there are unused variables) then
    v = Choose_at_random_a_unused_variable();
    Insert_unused_variable(v);
else
    t = U[0,1];
    if (t < 0.5 and the SHFS has only a SISO module) then
        Remove_one_used_variable();
    else
        Move_used_variable();
    end if
end if
```

- The insertion of an unused variable is as follows. Given an unused variable v, a module m is chosen randomly. Next, the operator decides if v is inserted into m with a probability of 0.5. If so, v is inserted as exogenous variable. Otherwise, if m has at least an exogenous variable as input, one of them, e, is selected at random. The mutation operator creates a new SISO module previous to m where the input is e and the output is v. In this case, v will be endogenous variable and input of module m. If m has not exogenous variables, the m's output is converted into a new exogenous variable of m and v will be the new output of m.

- To remove an used variable, first a module m of the SHFS is randomly chosen. If m has exogenous variables then one of them is removed at random. Otherwise, the module m is removed, thus linking the output of the module before m with the input of the module after m.

ISBN: 978-989-95079-6-8

- To move an used variable from a module to other, two modules of the chromosome are chosen at random: a source module m_1 where a variable is removed and a destination module m_2 where the variable is inserted. An exogenous variable of the m_1 is randomly removed and inserted as exogenous variable of the m_2. If there are not exogenous variables into the m_1 and consequently m_1 has one input and one output (a SISO module), its endogenous variable is chosen. In this situation, m_1 disappears and the chosen variable is inserted as an exogenous variable into m_2. If $m_1 = m_2$, an exogenous variable v is randomly selected, then a new module is created after m_1/m_2. The input of this new module (and therefore the output of the module m_1/m_2) is v while the output of the new module will be old output of m_1/m_2.

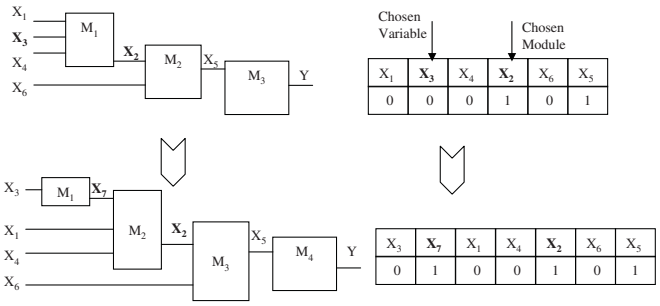

Figure 3: Example of insertion mutation of a unused variable (X_7) as endogenous variable, thus creating a new module

2.5 Fuzzy rule set learning

The learning fuzzy rule set of SHFS is as follows. Each module learns its fuzzy rule set with Mamdani rules depending on the inputs variables by WM [16]. The endogenous variables in SHFS are inferred by its respective module according to the input variables of the module. The obtained system is evaluated by training data examples, choosing the correspondent values of exogenous variables in that module. A module, according to learned fuzzy rule set, generates a value of output (an endogenous variable). The value is the next input of the module along with others exogenous variables. The membership functions are triangular shape distributed strong fuzzy partition.

2.6 Multi-objective approach

A generational approach with the multi-objective elitist replacement strategy of NSGA-II [17] is used. Crowding distance [17] in the objective function space is considered. Binary tournament selection based on the non-domination rank (or the crowding distance when both solutions belong to the same front) is applied. The crowding distance is normalized for each objective according to the extreme values of the solutions contained in the analyzed front.

2.7 Objective functions

We consider two objective functions to be minimize and so get a better precision and interpretability of the system.

2.7.1 Accuracy

It is computed as the RMSE:

$$F_1(S) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(S(x^i) - y^i)^2} \quad (1)$$

with S being the fuzzy system to be evaluated, N the data set size and (x^i, y^i) the ith input-output pair of the data set. The objective is to minimize this function to get good precision.

2.7.2 Interpretability

It is computed as follows:

$$F_2(S) = r(S) \quad (2)$$

with $r()$ being the number of rules of the fuzzy system.

Although the number of modules or the number of variables by module are notable criteria to evaluate the learned quality of the SHFS, the number of rules is more intuitive to validate the interpretability because it combines both criteria in one objective. If the number of rules is considered, the number of variables and modules is controlled automatically: as the number of rules decreases, the number of modules and variables increase.

3 Experimental Results

This section presents the experimental results. The objective of experimentation is to prove the reduction of number of rules and so, to assess the interpretability that can be compared to a referential learning method (WM in our case).

3.1 Problems

We have considered five regression problems with a moderate and high number of input real-valued variables: *Dee*[1], *Concrete*[2], *Elevators*[3], *Computer Activity (Comp-activ)*[3], and *Ailerons*[3].

Table 1 collects the main features of each problem, where *#InputVar* stands for number of input variables, *#Exam* for total number of examples, and *#LingTerms* for the number of triangular-shaped uniformly distributed linguistic terms considered for each variable in this experimental analysis.

Table 1: Data sets considered in the experimental analysis

Data set	*#InputVar*	*#Exam*	*#LingTerms*
Dee	6	365	5
Concrete	8	1030	5
Elevators	18	16559	3
Comp-activ	21	8192	3
Ailerons	40	13750	3

The experiments shown in this paper have been performed by a five fold cross validation with a total of 30 runs per problem (six runs for each data set partition). Thus, the data set is divided into five subsets of (approximately) equal size. The algorithm is then applied five times to each problem, each time

[1] KEEL: Knowledge extraction based on evolutionary learning. http://www.keel.es

[2] UCI Machine Learning Repository. Collection of regression datasets. http://archive.ics.uci.edu/ml/datasets.html

[3] L. Torgo, Collection of regression datasets. http://www.liacc.up.pt/ ltorgo/Regression/DataSets.html

ISBN: 978-989-95079-6-8

leaving out one of the subsets from training, but using only the omitted subset to compute the test error.

Our algorithm has been run with the following parameter values: 1,000 iterations, 60 as population size, 0.7 as crossover probability, and 0.2 as mutation probability per chromosome. We have not performed any previous analysis to fix these values, so better results may probably be obtained by tuning them though we have informally noticed our algorithm is not specially sensitive to any parameter.

3.2 Obtained results

For each problem we present the five validation test results (Table 2), where $RMSE_{tra}$ and $RMSE_{tst}$ are the approximation error (eq. 1) values over the training and test data set respectively, *#M*, *#R*, *#V* stand for the number of modules, the number of fuzzy rules, and the number of variables respectively.

Since our algorithm performs multi-objective optimization, several solutions are returned in each run. Therefore, we show five representative solutions from the final Pareto-optimal set: the first row of each problem is the best solution (highest precision), the second row is the first quartile, the third row is the median (it is not the average), the fourth row is the third quartile, and the fifth row is the worst solution (lowest precision).

Table 2: Obtained Results

	Method	$RMSE_{tra}$	$RMSE_{tst}$	*#M*	*#R*	*#V*
DEE	WM	0.375594	0.477303	1.0	178.4	6.0
	GSHFS	0.372482	0.474365	1.2	157.3	5.6
		0.396565	0.460321	1.8	95.6	5.2
		0.430456	0.484292	1.8	66.1	4.6
		0.495178	0.507092	2.1	28.6	3.7
		0.681968	0.695144	1.2	5.5	1.2
CONCRETE	WM	8.548268	9.822000	1.0	310.4	8.0
	GSHFS	8.479054	9.356946	1.4	257.1	7.2
		9.436752	9.812343	2.2	114.7	6.0
		10.700607	10.839647	2.3	53.3	5.2
		13.111270	13.450416	2.4	25.4	4.2
		16.669822	16.477185	1.3	5.6	1.3
ELEVATORS	WM	0.567771	0.542072	1.0	511.0	18.0
	GSHFS	0.487507	0.488641	2.3	39.7	7.7
		0.498203	0.498650	2.2	27.5	6.5
		0.517594	0.517673	2.4	18.9	5.7
		0.563252	0.563026	2.0	11.7	4.3
		1.421620	1.422627	1.6	3.8	1.8
COMP-ACTIV	WM	9.050474	9.084798	1.0	425.6	21.0
	GSHFS	5.398862	5.400107	2.5	37.5	8.1
		5.880516	5.874316	2.3	25.1	6.6
		6.708214	6.719212	2.2	16.8	5.5
		8.888890	8.935257	2.1	9.7	3.9
		31.900506	31.756379	1.7	4.1	2.0
AILERONS	WM	0.253971	0.255557	1.0	1080.6	40.0
	GSHFS	0.238855	0.239763	2.8	57.1	10.0
		0.254118	0.254958	2.7	37.7	8.2
		0.276632	0.277216	2.6	24.0	6.8
		0.318274	0.318159	2.3	12.9	5.0
		0.724993	0.731567	2.2	5.9	2.9

3.3 Analysis

The main objective is to obtain a fuzzy system with good tradeoffs between precision and interpretability. When the number of rules of a fuzzy system decreases, the system gets better interpretability, but the precision is lower and vice versa. This event can be observed clearly in the results obtained as problems with either low or high number of variables. Notice that the fuzzy rule set is learned individually for each module.

First set of Table 2 shows five solutions of the Pareto obtained by our algorithm in *Dee* problem. We can observe that the precision is high when the number of rules is high and, consequently, the fuzzy system has a lower interpretability. If we compare our algorithm with WM method, the best solution (highest precision) is better in precision, having a number of rules and a number of variables lower. In first quartile, the precision has a little increase according to solution obtained by WM in the $RMSE_{tra}$, but the $RMSE_{tst}$ is lower and the interpretability is decreased in more than a half.

However, if we observe the problems with a considerable number of variables (more than eight), our algorithm has better performance. Lets look at *Elevators* problem. It has 18 variables. The best solution (highest precision) obtained by our algorithm is better than WM solution. The $RMSE_{tra}$ and $RMSE_{tst}$ have been decreased. The interpretability is better because the SHFS has a lower number of variables and a higher number of modules. The number of rules has been decreased by 92%. The third quartile of the $RMSE_{tra}$ is better in precision than WM. The third quartile of the $RMSE_{tst}$ is a little worse but the number of rules obtained by our algorithm has decreased by 95%, 98%, and 96%, respectively.

In *Ailerons* problem (40 variables), we can see that our solution with highest precision is better than the solution obtained by WM. We want to emphasize that the number of rules is decreased by 95% and the number of variables by 75%. The median of the $RMSE_{tra}$ is a solution with a precision a little higher than WM solution, but the median of the $RMSE_{tst}$ is lower and the interpretability is better than the solution of WM, due to decrease of the number of variables and its distribution in modules.

To sum up, we have seen that in problems with a low moderate number of variables, the precision is worse when the number of rules decrease. It does not happen in problems with a higher number of variables because with a lower number of variables, the correlation between them is low, because they are important. Nevertheless, in problems with a higher number of variables exists more dependence between them: a hierarchical structure and a lower number of variables help to decrease the complexity of the learned fuzzy model.

Figure 4 shows the average Pareto front obtained in Ailerons problem by our algorithm. In it we can see that if there are many rules (lower interpretability), then the precision is higher. The SHFS gets a better interpretability with a lower precision.

Finally, we show an example of GSHFS hierarchical structure obtained by our algorithm in Figure 5. It is an example of non-dominated solution obtained by GSHFS in a data set partition of *Ailerons*. The obtained values are: $RMSE_{tra}$ = 0.214068, $RMSE_{tst}$ = 0.219682, #M = 3, #R = 63, #V = 9. The results obtained by WM in the same data set partition are: $RMSE_{tra}$ = 0.254144, $RMSE_{tst}$ = 0.261472, #M = 1, #R = 1072, #V = 40. We can observe that $RMSE_{tra}$ and $RMSE_{tst}$ obtained by our algorithm is lower than the values obtained by WM, decreasing the number of rules by 94%.

ISBN: 978-989-95079-6-8

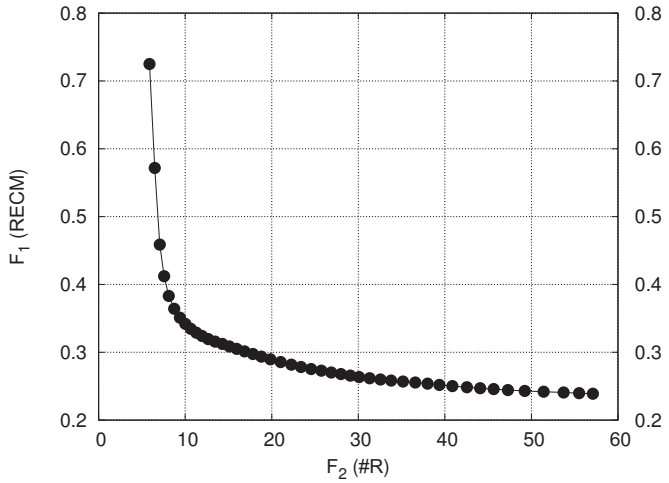

Figure 4: Average Pareto front obtained in Ailerons problem

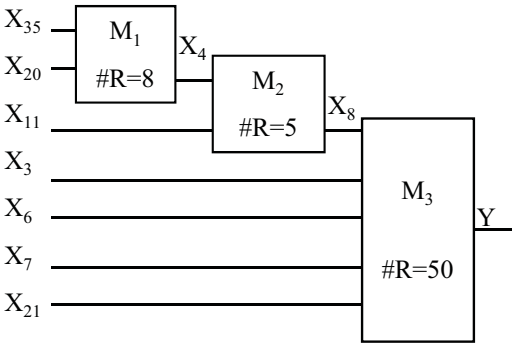

Figure 5: Example of a solution obtained by GSHFS in the *Ailerons* problem with #M = 3, #R = 63, #V = 9

4 Conclusion and Further Work

We have proposed a multi-objective algorithm applied to learning SHFS for palliate exponential increase of the number of rules when number of variables increases. The set of variables is divided into modules by the algorithm. We have proved that this division by SHFS can obtain good results in problems with a higher number of variables.

As further work, we suggest to add mechanisms for a better precision, for example, a global learning of fuzzy rule set, a detailed study of the interpretability of each rule of the fuzzy system, and to extend this algorithm for parallel and hybrid structure learning.

Acknowledgment

This work was supported in part by the Spanish Ministry of Science and Innovation (grant no. TIN2008-06681-C06-01), and by Andalusian Government (grant no. P07-TIC-3185).

References

[1] G.V.S. Raju, J. Zhou, and R.A. Kisner. Hierarchical fuzzy control. *International Journal Control*, 54(5):1201–1216, 1991.

[2] M.G. Joo and J.S. Lee. Hierarchical fuzzy control scheme using structured takagi-sugeno type fuzzy inference. In *Proceedings of IEEE International Fuzzy Systems Conference*, pages 78–83, Seoul, Korea, 1999.

[3] T.P. Hong and J.B. Chen. Finding relevant attributes and membership functions. *Fuzzy Sets and Systems*, 103(3):389–404, 1999.

[4] A. González and R. Pérez. Selection of relevant features in a fuzzy genetic learning algorithm. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 31(3):417–425, 2001.

[5] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions in Fuzzy Systems*, 3(3):260–270, 1995.

[6] T. Taniguchi, K. Tanaka, H. Ohtake, and H.O. Wang. Model construction, rule reduction, and robust compensation for generalized form of takagi-sugeno fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 9(4):525–538, 2001.

[7] D. Wang, X.J. Zeng, and J.A. Keane. Learning for hierarchical fuzzy systems based on gradient-descent method. In *Proceedings of IEEE International Conference on Fuzzy Systems*, pages 92–99, 2006.

[8] K. Shimojima, T. Fukuda, and Y. Hasegawa. Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *Fuzzy Sets and Systems*, 71:295–309, 1995.

[9] A.E. Gaweda and R. Scherer. Fuzzy number-based hierarchical fuzzy system. *Lecture Notes in Computer Science*, 3070:302–307, 2004.

[10] F. Cheong. A hierarchical fuzzy system with high input dimensions for forecasting foreign exchange rates. *IEEE Congress on Evolutionary Computation, CEC*, pages 1642–1647, 2007.

[11] S. Aja-Fernández and C. Alberola-López. Matriz modeling of hierarchical fuzzy systems. *IEEE Transactions in Fuzzy Systems*, 16(3):585–599, 2008.

[12] P. Salgado. Rule generation for hierarchical collaborative fuzzy system. *Applied Mathematical Modelling, Science Direct*, 32:1159–1178, 2008.

[13] V. Torra. A review of the construction of hierarchical fuzzy systems. *International Journal of Intelligent Systems*, 17:531–543, 2002.

[14] Y. Chen, J. Dong, and B. Yang. Automatic design of hierarchical ts-fs model using ant programming and pso algorithm. In C. Bussler and D. Fensel, editors, *Proceedings 12th International Conference on Artificial Intelligence, Methodology, Systems and Applications, Lecture Notes on Artificial Inteligence, LNAI 3192*, pages 285–294, 2004.

[15] T.M. Jelleli and A.M. Alimi. Improved hierarchical fuzzy control scheme. *Adaptive and Natural Computing*, 1:128–131, 2005.

[16] L.X. Wang and J. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427, 1992.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarevian. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

ISBN: 978-989-95079-6-8