

# Multi-objective genetic learning of serial hierarchical fuzzy systems for large-scale problems

Alicia D. Benítez · Jorge Casillas

© Springer-Verlag 2012

**Abstract** When we face a problem with a high number of variables using a standard fuzzy system, the number of rules increases exponentially and the obtained fuzzy system is scarcely interpretable. This problem can be handled by arranging the inputs in hierarchical ways. This paper presents a multi-objective genetic algorithm that learns serial hierarchical fuzzy systems with the aim of coping with the curse of dimensionality. By means of an experimental study, we have observed that our algorithm obtains good results in interpretability and accuracy 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 (FS) 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 (Raju et al. 1991; Joo and Lee 1999). Indeed, if we have  $n$  variables and  $k$  linguistic terms per variable, it requires up to  $k^n$  rules to build a complete Mamdani-type fuzzy system, and consequently, the accuracy-interpretability balance is broken. This problem is better known as the “curse of dimensionality.”

In order to solve it, several approaches have been suggested such as variable selection (Chiu 1996; Hong and Chen 1999; Jin 2000; González and Pérez 2001; Hong and Harris 2001; Lee et al. 2001; Xiong and Funk 2006; Tan et al. 2008) and rule set reduction (Ishibuchi et al. 1995; Taniguchi et al. 2001; Casillas et al. 2005; Alcalá et al. 2006, 2007, 2011b; Gacto et al. 2009). Nevertheless, when the number of variables increases considerably, this kind of reduction may not be enough to solve it. There is a different approach to deal with that: hierarchical fuzzy systems (HFSs). 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.

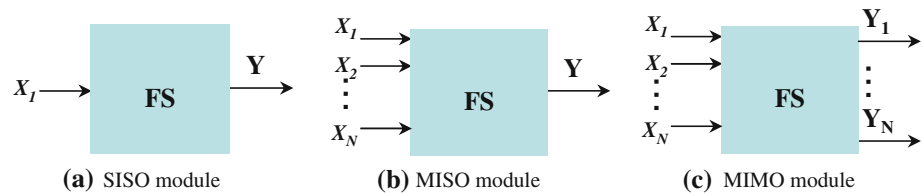
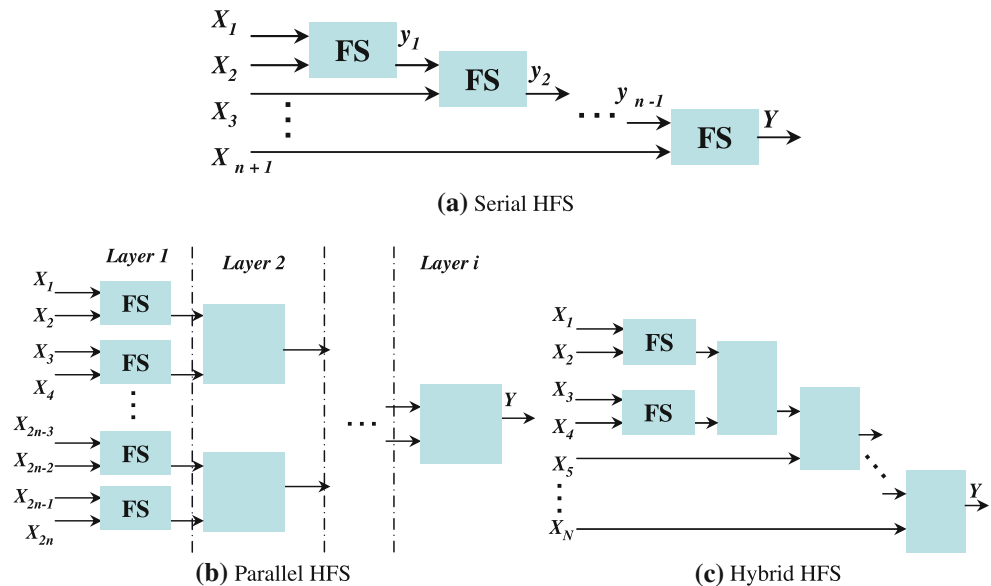
We may distinguish between three types of modules (Fig. 1):

- SISO (Single Input, Single Output): It has one input and one output (Fig. 1a).
- MISO (Multiple Inputs, Single Output): It has several inputs and a single output (Wang et al. 2006; Chen et al. 2007; Zajaczkowski and Verma 2012) (Fig. 1b). We can find Fuzzy Logic Unit (FLU) in this kind of modules. An FLU special case has two inputs and one output, which is equally found in the literature (Joo and Lee 1999, 2002; Shimojima et al. 1995; Wang 1998; Lee et al. 2003; Gaweda and Scherer 2004; Jelleli and Alimi 2005, 2010; Zhang and Zhang 2006; Cheong 2007; Aja-Fernández and Alberola-López 2008).

---

A. D. Benítez  
Calculation Centre, Astrophysics Institute of Andalusia (IAA),  
Spanish National Research Council (CSIC),  
18008 Granada, Spain  
e-mail: abenitez@iaa.es

J. Casillas (✉)  
Department of Computer Science and Artificial Intelligence,  
CITIC-UGR (Research Center on Information  
and Communication Technology), University of Granada,  
18071 Granada, Spain  
e-mail: casillas@decsai.ugr.es

**Fig. 1** Types of modules**Fig. 2** Types of hierarchical structures

- MIMO (Multiple Inputs, Multiple Outputs): They have several inputs and outputs (Salgado 2008) (Fig. 1c).

Apart from distinguishing several kinds of modules, it is also possible to find different types of hierarchical structures. There are different classifications (Gaweda and Scherer 2004; Duan and Chung 2002; Torra 2002), though the most general one is the following (Aja-Fernández and Alberola-López 2008):

- Serial HFS (SHFS): The input of a module is the output of the previous ones, along with external variables (Lee et al. 2003; Cheong 2007; Wang 1999; Aja-Fernández and Alberola-López 2008; Zeng et al. 2008; Benítez and Casillas 2009; Zajaczkowski and Verma 2012) (Fig. 2a).
- Parallel HFS (PHFS): This system is organized in layers (Fig. 2b). The first one is made up of a set of modules receiving the input variables. Each variable is used as an input only in a single module. The outputs of the modules in the first layer are the inputs 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 (Holve 1998; Joo and Lee 1999; Jelleli and Alimi 2005, 2010; Lee et al. 2003; Salgado 2008; Zhang and Zhang 2006;

Aja-Fernández and Alberola-López 2008; Joo and Sudkamp 2009).

- Hybrid HFS (HHFS): This type of HFS is a mixture of the two previous ones (Joo and Lee 2002; Wang et al. 2006; Chen et al. 2004, 2007; Aja-Fernández and Alberola-López 2008) (Fig. 2c).

Figure 2 shows this classification of hierarchical structures.

We want to emphasize each module of an HFS deals with a subset of variables, in other words, it does not handle the problem as a whole. Thanks to the decomposition of the FS made in an HFS, the complexity of each module is significantly reduced due to the fact that the modules' rules are simpler than the rules of the FS with a module since the number of variables of each module is lower.

Several approaches have studied the best way to deal with the linking variables used to link modules. The general way is to generate new linking variables without linguistic meanings (Lee et al. 2003; Cheong 2007; Wang 1999; Zeng et al. 2008; Holve 1998; Joo and Lee 1999, 2002; Jelleli and Alimi 2005, 2010; Salgado 2008; Zhang and Zhang 2006; Joo and Sudkamp 2009; Wang et al. 2006; Chen et al. 2004, 2007; Aja-Fernández and Alberola-López 2008; Zajaczkowski and Verma 2012). Gaweda and Scherer (2004) approach this task with fuzzy numbers.

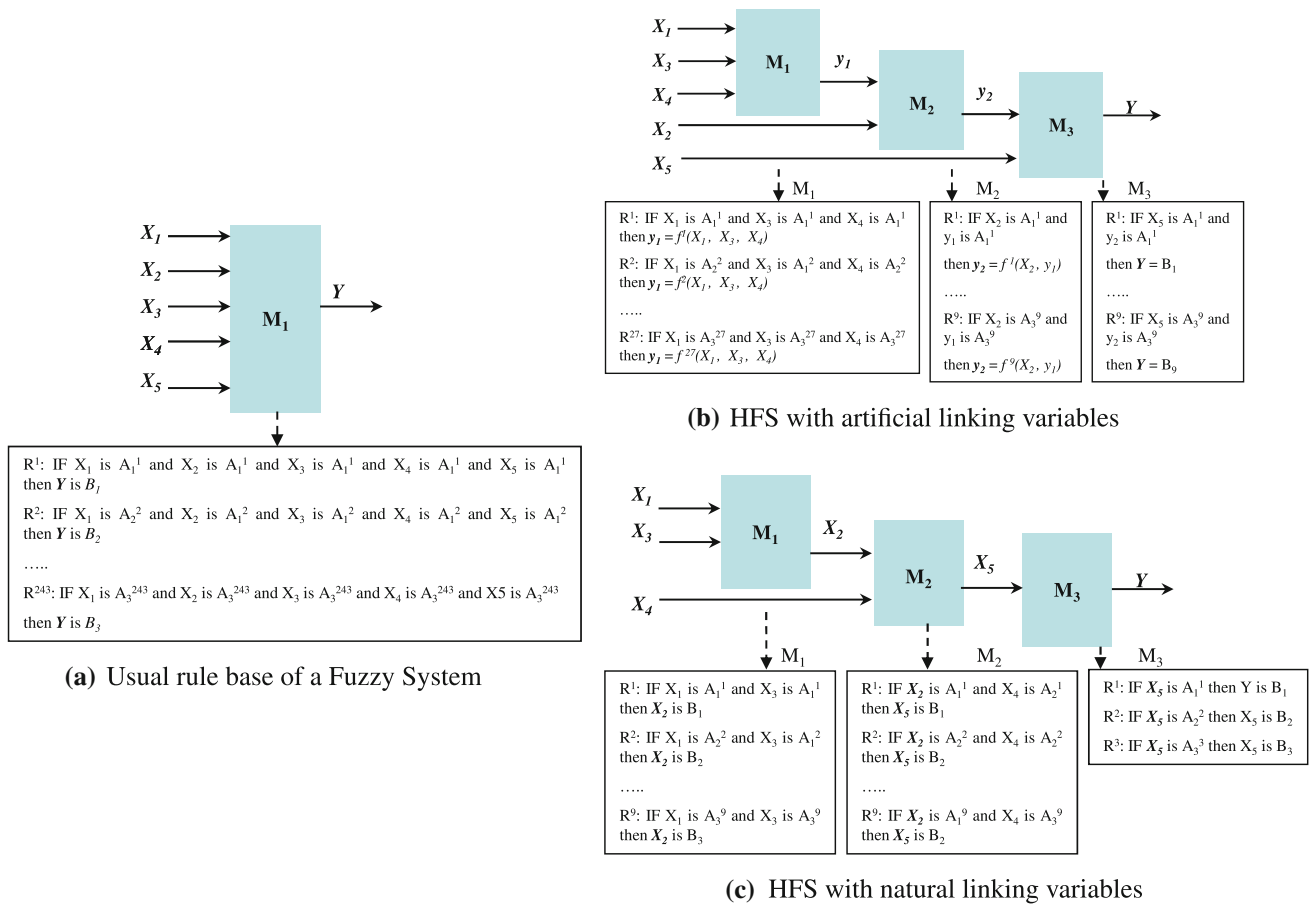


Fig. 3 Types of rule bases in a fuzzy system

In this paper, the term mutual subsethood appears as a similarity measure based on the degree of inclusion of  $A'$  in  $A$  and, at the same time, on the degree of inclusion of  $A$  in  $A'$ ,  $A$  being a fuzzy set and  $A'$  a fuzzy numbers set.

Jelleli and Alimi (2005) propose an iterative algorithm, which associates linearly the inputs in pairs  $(x_{L,i}, x_{L,i+1})$ , building a first layer with a set of FLUs, with  $L$  being  $L$ th the layer and  $i$  the  $i$ th FLU. Each FLU generates a linking output,  $y_{L,i}$ , and they are associated in pairs  $(y_{L,i}, y_{L,i+1})$ . The non-dominant linking output,  $d_{L,i}$ , is selected from each pair. Next, these variables are the inputs of the next layer. This procedure is a looper manner until sweeping all hierarchical levels.

Therefore, we distinguish three types of rule bases: (1) a usual rule base of an FS (Fig. 3a), (2) HFS with artificial linking variables (the linking variables are new variables created by means of a mathematic function; in Fig. 3b, variables  $y_1$  and  $y_2$ ), and (3) HFS with natural linking variables (the linking variables are problem's variables; in Fig. 3c, variables  $X_2$  and  $X_5$ ). Let the FSs be in Fig. 3, if we have five variables and three linguistic terms per variable, the FS in Fig. 3a requires up to  $3^5 = 243$  rules to build a

complete FS; in Fig. 3b there are 45 rules ( $3^3$  rules in  $M_1$ ,  $3^2$  rules in  $M_2$ , and  $3^2$  rules in  $M_3$ ), and in Fig. 3c there are 21 rules ( $3^2$  rules in  $M_1$ ,  $3^2$  rules in  $M_2$ , and  $3^1$  rules in  $M_3$ ). In these figures, we can observe that the lowest total number of rules is obtained in Fig. 3c. If we compare the FS in Fig. 3c with the FS in Fig. 3a, we can notice the variables distributed in modules in an HFS decrease the total number of rules and, moreover, the complexity of each rule is reduced. If we compare the FS in Fig. 3c with the FS in Fig. 3b, the number of rules in Fig. 3c is lower than Fig. 3b because it does not generate artificial linking variables.

Therefore, in this paper, we suggest the use of the problem's variables in order to communicate the modules. A previous version of our proposal was published in Benftez and Casillas (2009). Each module has a set of Mamdani-based rules, which infer the linking variables. The FS generated is more interpretable because it uses Mamdani-type rules and the natural linking variables have semantic meaning. As far as we know, it is the first time that an algorithm combines an HFS with Mamdani-type rules. Besides, we use a robust approach with a learning

process based on evolutionary computation, called genetic fuzzy systems (GFS). GFS combines a population-based search with a linguistic representation interpretable (Casillas and Carse 2009; Nojima et al. 2011).

We propose a multi-objective genetic algorithm (Multi-Objective GA) with the following features:

- It allows large-scale problems to be dealt, thanks to its hierarchical structure and the inclusion of variable selection.
- It guarantees good interpretability due to the algorithm uses of problem's variables in order to link modules. The hierarchical structure decreases the number of rules and each rule is simpler because the number of variables per rule is lower. Moreover, it allows the algorithm to deal with Mamdani-type rules for a better understanding.
- It obtains different trade-offs between interpretability and accuracy, thanks to the multi-objective algorithm, and so, it generates different solutions with different trade-offs.

Most HFS proposals are focused on improving the accuracy. However, the aim of our proposal is to improve the interpretability by means of a precise and compact set of rules, while the accuracy is maintained or improved.

The paper is organized in the following sections: in Sect. 2, a summary of the most relevant algorithms that work in this area is carried out; in Sect. 3, the proposed algorithm is described: structure, coding scheme, initialization, genetic operators, inference mechanism, rule base learning, multi-objective approach, and the objective functions; in Sect. 4, the empirical study is shown: the problems used are described, the results obtained, and an analysis and study of its behavior; in Sect. 5, a comparison between our proposal and the state of the art in the area of GFSs is provided; in Sect. 6, conclusion and future work are discussed.

## 2 Related works

In the literature, there are some proposals to deal with HFS without learning a hierarchical structure (Wang 1998, 1999; Joo and Lee 1999, 2002; Gaweda and Scherer 2004; Lee et al. 2003; Zhang and Zhang 2006; Cheong 2007; Zajackowski and Verma 2012). These cases are outside of our interest. Indeed, the design of an optimal hierarchical structure is as important as the accuracy of each module because if the HFS has a higher number of modules, the number of errors carried by the system will be higher too. This section reviews the main existing

approaches to learn hierarchical structures. The description is sorted in chronological order, and does not mention approaches that use hierarchical structures, but do not learn them.

The algorithm of Shimojima et al. (1995) learnt a hybrid hierarchical structure with a GA and back-propagation method. It uses exponential radial basis membership functions (MFs) and are tuned with supervised learning by means of the gradient descent method. The rules of the HHFS are Takagi–Sugeno–Kang (TSK).

Chen et al. (2004) designed an algorithm with the capacity to learn HHFS structures by means of ant programming and realizes a fine tuning of rule's parameters using Particle Swarm Optimization (PSO) algorithm. It codes the HFS in a tree structure and uses TSK rules with exponential MFs.

Wang et al. (2006) used the descent gradient method in order to learn the HHFS structure. It uses Mamdani-type rules and triangular MFs.

Chen et al. (2007) improved the version of Chen et al. (2004), which learns HHFS structures by means of Probabilistic Incremental Program Evolution (PIPE) and realizes a fine tuning of the rule's parameters using evolutionary programming. The shapes of MFs are learnt with adaptive techniques and it uses TSK rules.

Aja-Fernández and Alberola-López (2008) create a new way of learning the rule base by means of a matrix procedure called Fast Inference using Transition Matrices (FITM), which is proposed as a methodology to perform inferences with the standard additive model. It uses Mamdani-type rules and this rule base learning can be used in SHFS, PHFS, and HHFS.

Joo and Sudkamp (2009) designed a two-layer PHFS. It uses Mamdani-type rules in the first layer, TSK-type rules in the second layer, and applies a rule reduction establishing a dependence relationship among first-layer FSs. The MFs are triangular. This approach is described in detail in Appendix 3.

The learning of the HFS problem remains an open problem. For example, Jelleli and Alimi (2010) have recently published an learning algorithm of PHFS. The variables are grouped in modules by means of the learning the fuzzy behavior from data. Later, a decision maker chooses the modules of the next layer according to a weight. The MFs are Gaussian and TSK rules are used.

Table 1 shows a summary of the approaches with hierarchical structure learning. Notice that in all the reviewed algorithms, new variables are generated to link the modules. However, as stated above, our algorithm will use natural variables, thus ensuring better interpretability.

**Table 1** Summary of the main existing approaches with hierarchical structure learning

Author	Membership functions	Type of rules	Structure	Linking variable	Learning
Shimojima et al. (1995)	Radial basis	TSK	HHFS	Artificial	GA, back-propagation
Chen et al. (2004)	Exponential	TSK	HHFS	Artificial	Ant programming
Chen et al. (2007)	Exponential	TSK	HHFS, SHFS	Artificial	PIPE
Aja-Fernández et al. (2008)	Triangular	Mamdani	PHFS, HHFS	Artificial	FITM
Joo and Sudkamp (2009)	Triangular	TSK	PHFS	Artificial	Greedy
Jelleli and Alimi (2010)	Gaussian	TSK	PHFS	Artificial	Decision maker
Our proposal (2012)	Triangular	Mamdani	SHFS	Natural	Multi-objective GA

### 3 GSHFS algorithm

The algorithm, called Genetic SHFS (GSHFS), uses a multi-objective genetic algorithm where there are two objectives to minimize: the mean squared error (MSE) and the maximum number of rules. Our algorithm has three specific genetic operators: one crossover operator and two mutation operators. Its scheme is shown in Algorithm 1. The following subsections detail the different components of the algorithm.

**Algorithm 1** GSHFS Algorithm

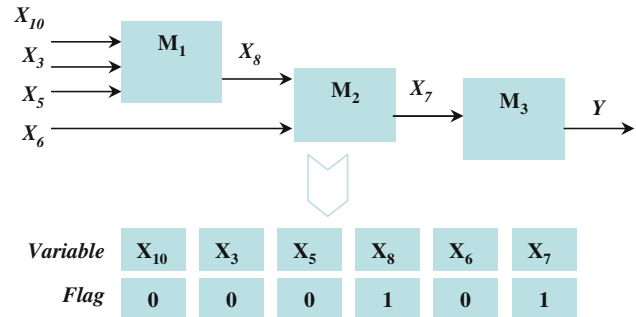
**Require:** Population size, crossover and mutation probability.  
 Data set:  $D = \{(x, y) | x \in \mathbb{R}^n, y \in \mathbb{R}^m\}$ .  
 MF definitions.  
**Ensure:** Set of non-dominated solutions, each one with a different number of rules/accuracy tradeoff.  
 Initialization(P);  
 Evaluation(P);  
**while** (non stop condition) **do**  
   P1 ← Multiobjective\_Selection(P);  
   P2 ← Crossover(P1);  
   P3 ← Exchange\_Mutation(P2);  
   P4 ← Insert\_Mutation(P3);  
   Evaluation(P4);  
   P ← Multi-objective\_Replacement(P4);  
**end while**

#### 3.1 Coding scheme

We will distinguish between two kinds of input variables: (1) endogenous variable: it is a variable linking two modules, (2) exogenous variable: it is a system’s external variable.

In our algorithm, each individual represents an SHFS. The variable-length coding scheme 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).

A variable with ‘1’ in the flag field means that is an endogenous variable and a module exists. This variable is the input of the next module. All the used 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 variables. The highest hierarchy level



**Fig. 4** Example of coding scheme

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.

Figure 4 shows an example of the coding scheme. This example of SHFS has ten inputs: four of them are exogenous variables ( $X_{10}, X_3, X_5,$  and  $X_6$ ) and two of them are endogenous variables ( $X_8$  and  $X_7$ ). There are four variables ( $X_1, X_2, X_4,$  and  $X_9$ ) that does not appear.

#### 3.2 Initialization

The algorithm generates an initial population randomly. The variable index and the endogenous/exogenous flag is randomly chosen for each gene. The only restriction is that the variable cannot be repeated and the flag of the first gene cannot be ‘1’ (i.e., the first variable has to be exogenous). The algorithm is constrained to a maximum number of modules because if the number of modules is high, the propagation error will increase considerably. On the other hand, we want to reduce the search space too.

#### 3.3 Crossover operator

The crossover operator is applied according to a probability between mated parents and exploits the search space. When the crossover operator is applied to two parents,  $P_1$  and  $P_2$ , different cases may be distinguished: (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 (or the reverse case with  $P_2$  and  $P_1$ ), (3) both parents,  $P_1$  and  $P_2$ , have one module. A parent-centric approach has been used in some cases where the offspring mainly inherits the information of one of the parents and takes the secondary parent in order to add diversity. Figure 5 shows the cases by means of decision trees. The crossover operator controls the maximum number of modules in order to reduce the search space. It creates a new solution if the cross between two parents generates a number of modules less than or equal to the maximum number of modules.

The crossover is applied to individuals depending on the types of variables that the two parents have in common. Each case has an ordered priority list, thus the crossover is applied to the first matched.

### 3.3.1 Both parents, $P_1$ and $P_2$ , have several modules

- **Priority 1. Endogenous–Endogenous case** If  $P_1$  has common endogenous variables with  $P_2$ , a common endogenous variable is selected at random. This variable is the crossing 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 of the chromosome to the crossing point, but excluding it. The remaining one 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 6a illustrates an example of this case.
- **Priority 2. Endogenous–Exogenous case** If  $P_1$  has endogenous variables which are exogenous in  $P_2$ , a common variable is randomly selected as the crossing 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 of this variable are removed. This kind of alteration produces a lower aggressive change because the offsprings resemble their parents.
- **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 the crossing point. The offspring  $O_1$  is created centered on  $P_1$ , but excluding the crossing point. The first part of  $O_1$  is inherited from the first part of  $P_2$  (from the first gene to the crossing point). The repeated variables are removed in  $O_1$  from the part inherited from  $P_2$ .
- **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 other variables, is the input in a module that 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 to  $P_1$  and  $P_2$ . Later, the repeated variables of  $O_1$  in the part inherited from  $P_1$  are removed. If this case is true for  $P_2$ ,  $O_2$  offspring will be generated with the same procedure, but centered on  $P_2$ . Figure 6b shows an example of this case. The generated SHFS offspring has four modules. 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 no endogenous variable as the output since it coincides with the output of the SHFS.

- **Priority 5. Different variables case** If the variables in  $P_1$  and  $P_2$  are different, an endogenous variable from  $P_1$  and  $P_2$  is chosen at random as the crossing point. Offsprings are generated in the same way as in priority 1.

### 3.3.2 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 as in the Endogenous–Exogenous case when  $P_1$  and  $P_2$  have several modules. Figure 6c depicts this case.
- **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 is shown in Fig. 6d.
- **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 into the module of  $O_1$  with the output  $Y$ .

### 3.3.3 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 as in the Exogenous–Endogenous case when  $P_1$  and  $P_2$  have several modules. Figure 6e illustrates this.
- **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 as in the

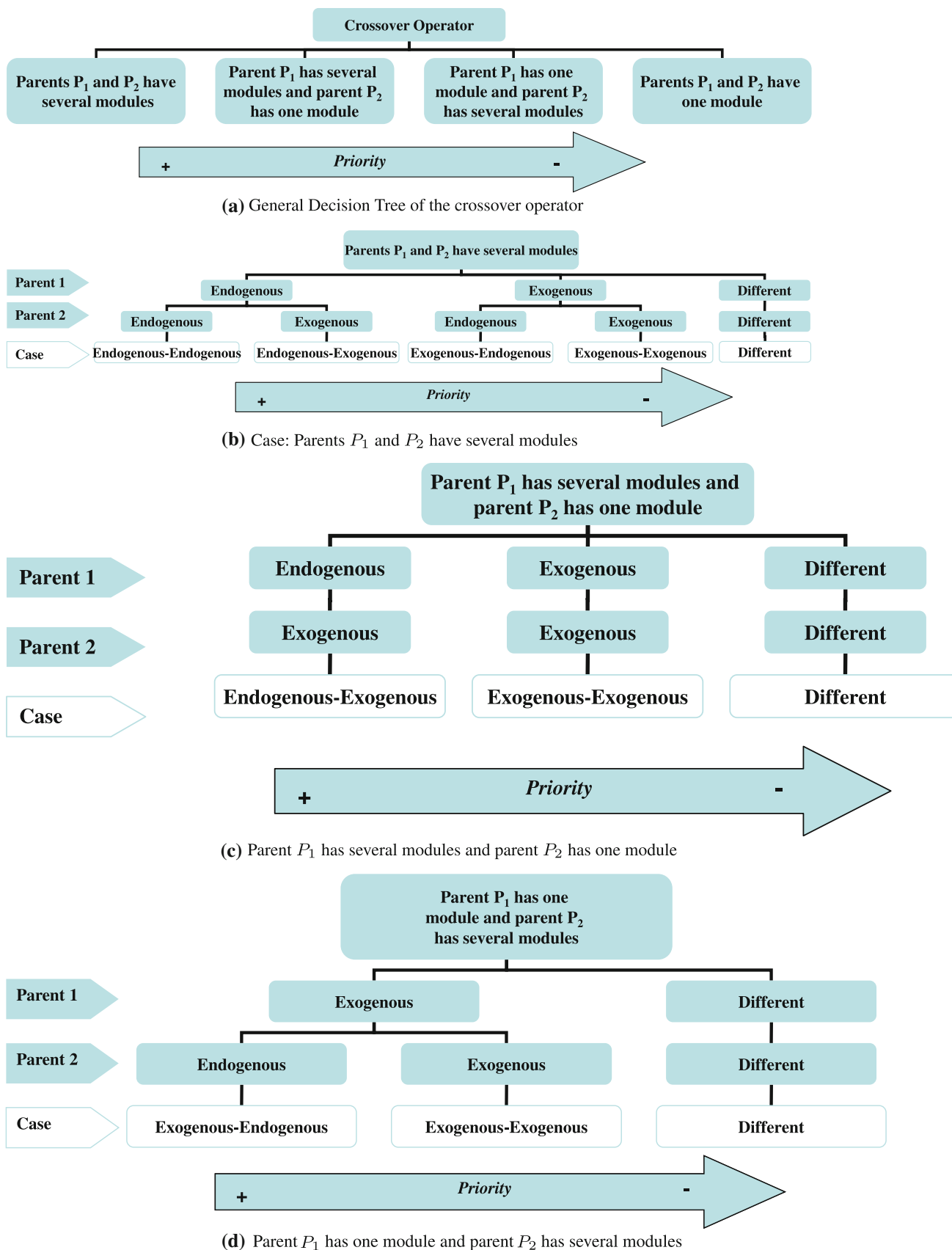
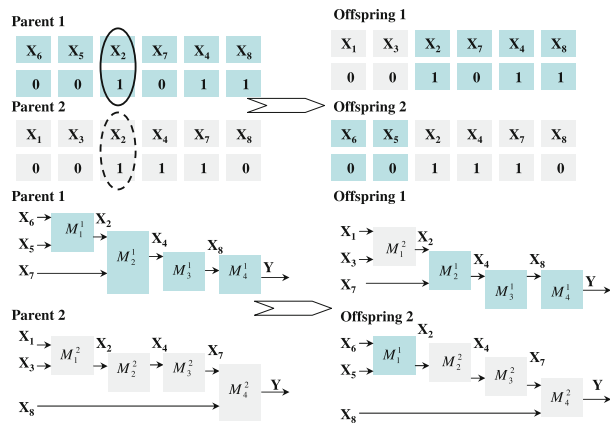
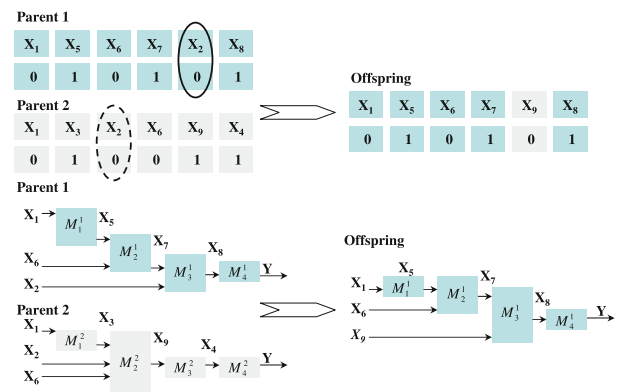


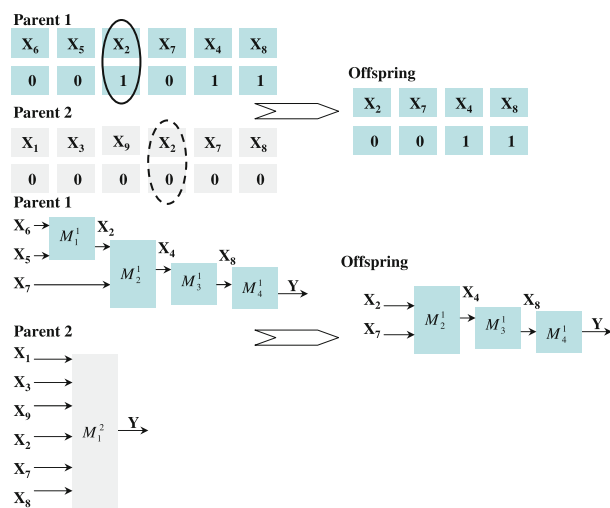
Fig. 5 Decision trees of the crossover operator



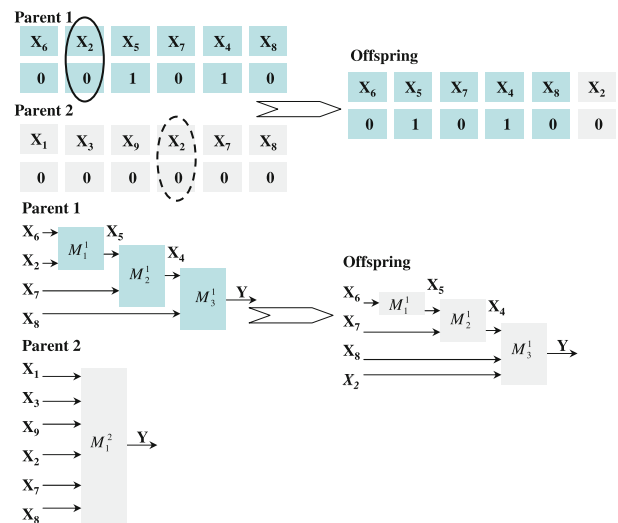
(a) Parents  $P_1$  and  $P_2$  have several modules: Endogenous-Endogenous case



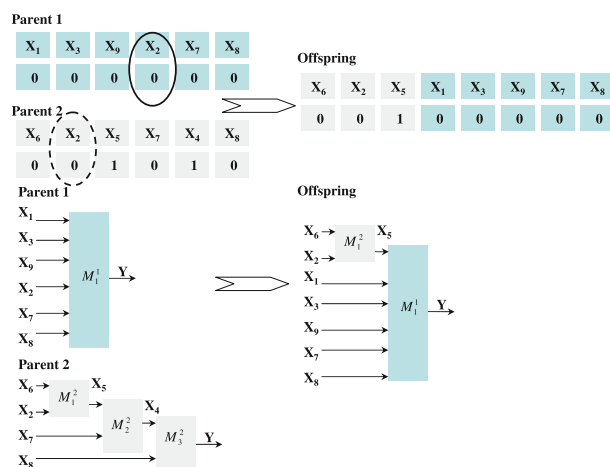
(b) Parents  $P_1$  and  $P_2$  have several modules: Exogenous-Exogenous case



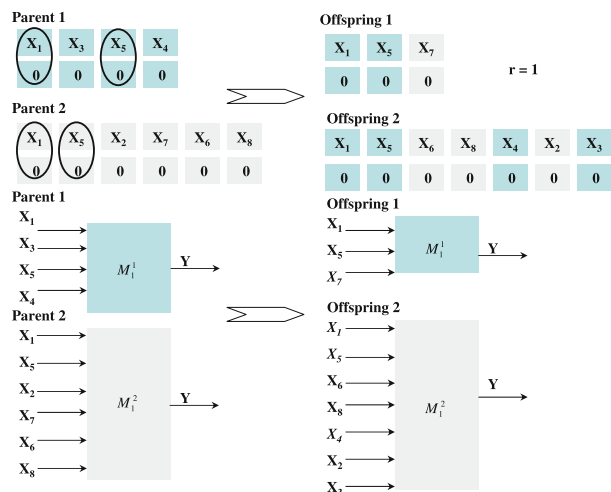
(c) Parent  $P_1$  has several modules and parent  $P_2$  has one module: Endogenous-Exogenous case



(d) Parent  $P_1$  has several modules and parent  $P_2$  has one module: Exogenous-Exogenous case



(e) Parent  $P_1$  has one module and parent  $P_2$  has several modules: Exogenous-Exogenous case



(f) Parents  $P_1$  and  $P_2$  have one module

Fig. 6 Crossover operator



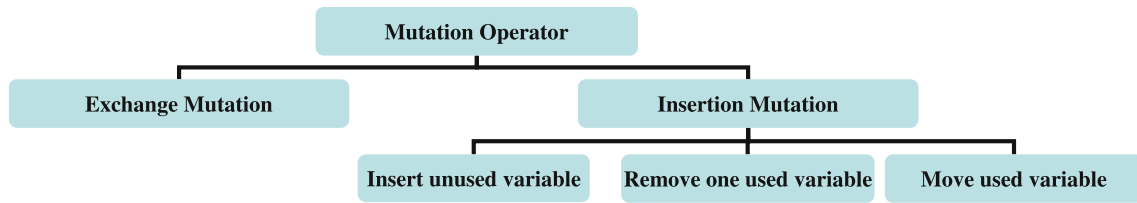
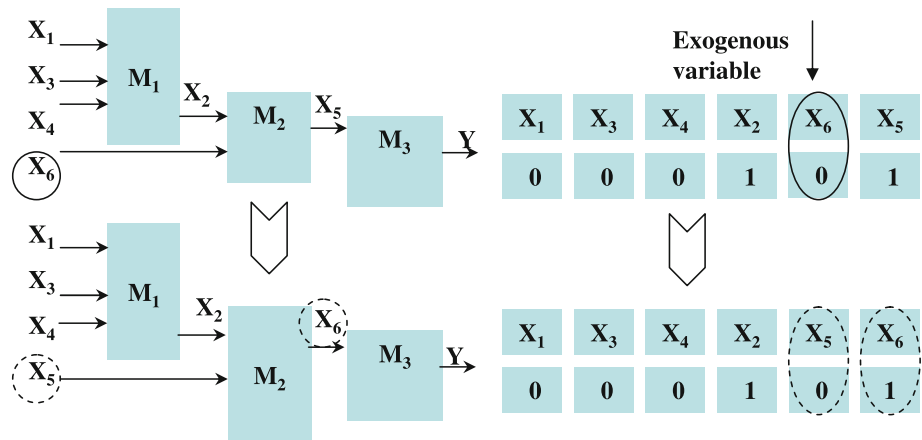


Fig. 7 Decision tree of the mutation operator

Fig. 8 Example of exchange mutation



Exogenous–Exogenous case when  $P_1$  and  $P_2$  have several modules.

- **Priority 3. Different variables case** If the 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 crossing point. The offsprings are generated in the same way as in priority 1.

### 3.4.1 Exchange mutation

This operator makes an exchange of variables in a module. It chooses an exogenous variable at random in a module and exchanges this exogenous variable for the endogenous variable of the module. Figure 8 shows an example.

### 3.4.2 Insertion mutation

The mutation operator distinguishes between used and unused variables in the individual. In accordance with this and by means of a probabilistic decision (Algorithm 2 shows a scheme), the mutation operator chooses between inserting an unused variable into a module, removing a used variable, or moving a used variable to the other module.

### 3.3.4 Both parents, $P_1$ and $P_2$ , have one module

In this case, the offsprings  $O_1$  and  $O_2$  are generated as follows (Fig. 6f shows this case): (1) the common exogenous variables of  $P_1$  and  $P_2$  are inserted into both offsprings, (2) the rest of the variables (no common variables between parents) are inserted in a set  $V$  with  $n = |V|$  being its size, (3) a number  $r$  is randomly generated: if there are no common variables between  $P_1$  and  $P_2$  then  $r \in \{1, \dots, n - 1\}$ ; otherwise,  $r \in \{0, \dots, n\}$ , (4)  $r$  variables are randomly taken out and inserted into  $O_1$ , (5) the rest of the variables are inserted into  $O_2$ .

## 3.4 Mutation operator

The mutation operator makes local changes in the chromosome. As well as the crossover operator, this operator controls the maximum number of modules. Two kinds of mutation have been designed (Fig. 7 shows a decision tree of this operator):

---

#### Algorithm 2 Insertion Mutation

---

```

r = U[0,1]; {Uniform probability}
if (r < 0.5 and there are unused variables) then
    v = Choose_at_random_a_unused_variable;
    Insert_unused_variable(v); {Algorithm 3}
else
    t = U[0,1];
    if (t < 0.5 and the SHFS has not only a SISO module) then
        Remove_used_variable; {Algorithm 4}
    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 an exogenous variable. Otherwise, if  $m$  has at least one exogenous variable as an 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  $v$ . Figure 9a shows an example. In this case,  $v$  will be endogenous and the input of module  $m$ . If  $m$  has no exogenous variables (Fig. 9b), the  $m$ 's output is converted into a new exogenous variable of  $m$  and  $v$  will be the new output of  $m$ . The mutation is applied if the solution has a number of modules less than the maximum number of modules.
- To remove a 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. Figure 9c depicts this case. Otherwise, the module  $m$  is removed, thus linking the output of the module before  $m$  with the input of the module after  $m$ . Figure 9d illustrates an example.
- To move a used variable from one module to another, 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 the exogenous variable of the  $m_2$ . If there are no 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 the old output of  $m_1/m_2$ . In this case, the mutation is applied if the solution has a number of modules less than the maximum number of modules.

Algorithms 3, 4, and 5 show the behavior of this mutation operator.

---

#### Algorithm 3 Insertion of an unused variable

---

**Require:** An unused variable:  $v$

```

1:  $m = \text{Choose\_at\_random\_a\_module}$ ;
2:  $r = \text{U}[0,1]$ ; {Uniform probability}
3: if ( $r < 0.5$ ) then
4:    $\text{Insert\_exogenous\_variable}(v, m)$ ;
5: else
6:    $t = \text{U}[0,1]$ ;
7:   if ( $t < 0.5$  and  $m$  is not a SISO module) then
8:      $e = \text{Choose\_at\_random\_an\_exogenous\_variable}(m)$ ;
9:      $\text{Create\_previous\_SISO\_module}(e, v, m)$ ;
10:  else
11:     $\text{Convert\_output\_variable\_to\_exogenous}(m)$ ;
12:     $\text{Convert\_variable\_to\_output}(v, m)$ ;
13:  end if
14: end if

```

---



---

#### Algorithm 4 Remove an used variable

---

```

1:  $m = \text{Choose\_at\_random\_a\_module}$ ;
2: if ( $m$  is not a SISO module) then
3:    $e = \text{Choose\_at\_random\_an\_exogenous\_variable}(m)$ ;
4:    $\text{Remove\_exogenous\_variable}(e, m)$ ;
5: else
6:    $\text{Remove\_module}(m)$ ;
7:    $\text{Link\_modules}(m - 1, m + 1)$ ;
8: end if

```

---



---

#### Algorithm 5 Move an used variable

---

```

1:  $m_1 = \text{Choose\_at\_random\_a\_module}$ ;
2:  $m_2 = \text{Choose\_at\_random\_a\_module}$ ;
3: if ( $m_1 \neq m_2$ ) then
4:   if ( $m_1$  is not a SISO module) then
5:      $e = \text{Choose\_at\_random\_an\_exogenous\_variable}(m_1)$ ;
6:      $\text{Remove\_variable}(e, m_1)$ ;
7:      $\text{Insert\_variable}(e, m_2)$ ;
8:   else
9:      $v = \text{Choose\_endogenous\_variable}(m_1)$ ;
10:     $\text{Remove\_module}(m_1)$ ;
11:     $\text{Insert\_exogenous\_variable}(v, m_2)$ ;
12:   end if
13: else
14:    $v = \text{Choose\_exogenous\_variable}(m_1)$ ;
15:    $w = \text{Choose\_endogenous\_variable}(m_1)$ ;
16:    $m_{\text{new}} = \text{Create\_new\_module\_after}(m_1)$ ;
17:    $\text{Set\_input\_module}(v, m_{\text{new}})$ ;
18:    $\text{Set\_output\_module}(w, m_{\text{new}})$ ;
19:    $\text{Link\_modules}(m_1, m_{\text{new}})$ ;
20: end if

```

---

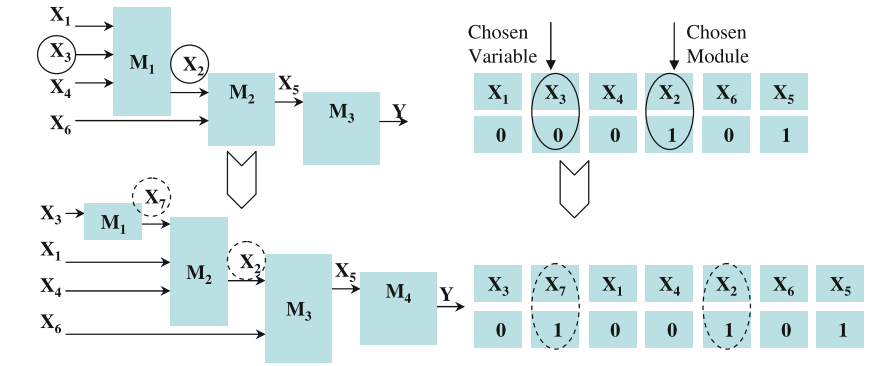
### 3.5 Inference mechanism

We consider the Max–Min inference scheme (i.e., T-conorm of the maximum as aggregation and T-norm of the minimum as relational operator), and the T-norm of the minimum as conjunction, and center-of-gravity as defuzzification. The inference in the SHFS is produced from left modules to right modules. When a module infers the output, it produces an error that is carried in the SHFS. For this reason, it is not good to have a high number of modules in an HFS because the carried error will increase considerably. We must be careful to design the system in order to get a lower propagation of the error because the more modules are considered, the more uncertainty we get (Maeda 1996).

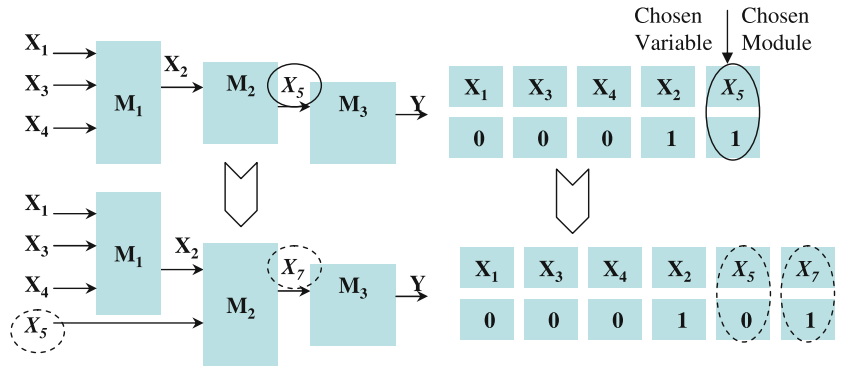
### 3.6 Fuzzy rule set learning

The fuzzy rule set learning is as follows: each module learns its Mamdani-type fuzzy rule set by Wang–Mendel (WM) method (Wang and Mendel 1992) (see Appendix 1). The module's exogenous variables are extracted from the data set and the endogenous variables in SHFS are inferred by its respective module according to the input variables of

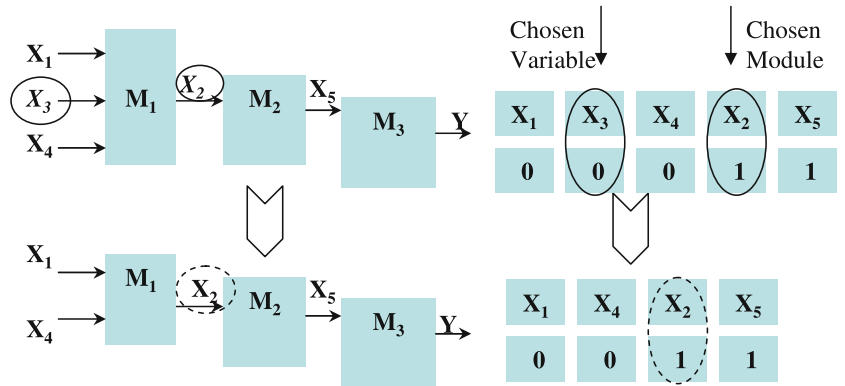
**Fig. 9** Insertion mutation operator



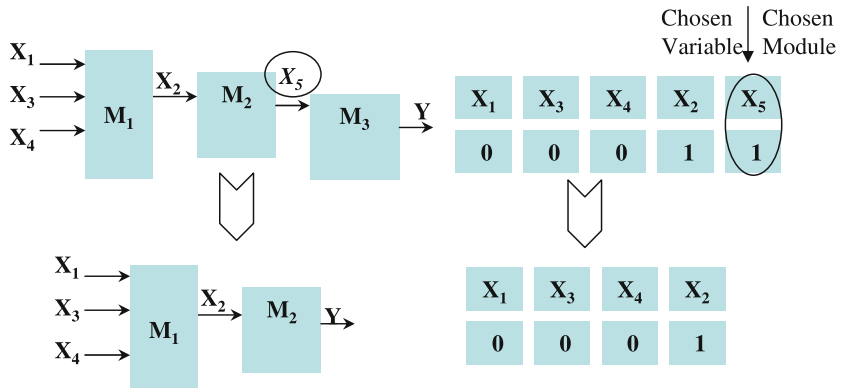
(a) Insertion of an unused variable. Case: If  $m$  has at least an exogenous variable as input



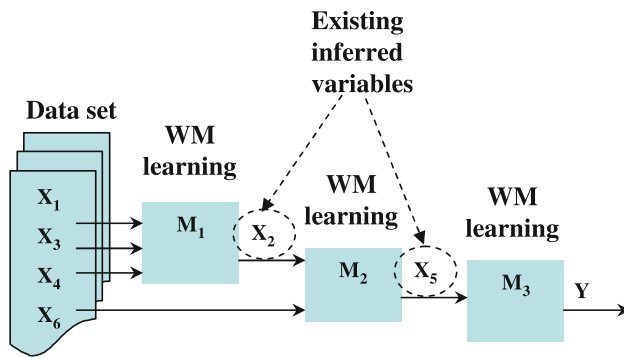
(b) Insertion of an unused variable. Case: If  $m$  has not exogenous variables



(c) Remove an used variable. Case: If  $m$  has exogenous variables



(d) Remove an used variable. Case: If  $m$  has not exogenous



**Fig. 10** Example of inference process

the module. Figure 10 shows a graphic example: the first module receives three exogenous variables from the data set and infers an endogenous natural variable; the value of the endogenous inferred variable (it is not extracted from the data set) is the input of the next module along with other exogenous variables extracted from the data set, and so on. The obtained system is evaluated by training data examples and choosing the corresponding values of exogenous variables in that module. The aim is to build a closed system in order to ensure good interpretability and reduce the error.

### 3.7 Multi-objective approach

A generational approach with the multi-objective elitist replacement strategy of NSGA-II (Deb et al. 2002) is used. Crowding distance 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.

We use NSGA-II because of its efficient and effective performance. Further analysis of the best multi-objective approach is away of the scope of this paper.

### 3.8 Objective functions

We consider two objective functions to minimize:

- *Mean square error* (MSE) If MSE is lower, the accuracy will be better. It is computed as:

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

with  $S$  being the SHFS to be evaluated,  $N$  the data set size and  $(x^i, y^i)$  the  $i$ th input–output pair of the data set. The objective is to minimize this function to get good accuracy.

- *Maximum number of rules* The system will be more interpretable with a lower maximum number of rules. It is computed as follows:

$$F_2(S) = \sum_{i=1}^M (k^n) \quad (2)$$

with  $M$  being the number of modules of the SHFS, and  $k$  and  $n$  the number of labels and variables per module, respectively. This objective has the main advantage (compared to considering, e.g., the final number of rules) of depending exclusively of the “static” structure of the SHFS, that is the number of modules, number of variables per module and number of linguistic terms used in each variable. Therefore, the objective is not influenced by the learning algorithm used to generate the fuzzy rule set at each module. This is especially relevant when the MFs are tuned, as two solutions with exactly the same hierarchical structure would generate different number of rules. Besides, although the number of modules or the number of variables per module is also notable criterion to evaluate the compactness of the SHFS, the maximum number of rules is a more intuitive criteria to validate interpretability because it combines both criteria in one objective. If the maximum number of rules is considered, the number of variables and modules will be controlled automatically; as the maximum number of rules decreases, the number of modules increases and the number of variables decreases. We will analyze this behavior in Sect. 4.4.

## 4 Experimental results

### 4.1 Problems

We have considered 14 regression problems with a moderate and high number of real-valued variables. Table 2 collects the main features of each problem, where *#InputVar* stands for the number of input variables, *#Exam* for the total number of examples, *#LingTerms* for the number of triangular-shaped uniformly distributed linguistic terms considered for each variable in this experimental analysis, and *Available* indicates the web site where the data sets are available. We set three labels per variable for large-scale problems because if the number of inputs and labels is high, the rule set will grow up considerably and the FS has lower interpretability. For this reason, a higher number of labels are used in problems with a lower number of inputs.

Our proposal solves regression problems. The data set with the highest number of inputs we have found of the considered web sites (Table 2, column *Available*) has been “Ailerons” with 40 real-valued variables. A data set called

**Table 2** Data sets considered in the experimental analysis

Data set	#InputVar	#Exam	#LingTerms	Available
Ele2	4	1,066	5	KEEL
Laser	4	993	5	KEEL
Dee	6	365	5	KEEL
Concrete	8	1,030	5	UCI
Census-House 8L (Census 8L)	8	22,784	5	Delve
Census-House 8H (Census 8H)	8	22,784	5	Delve
Weather-Ankara (WeAnk)	9	1,609	5	KEEL
Mortgage	15	1,049	5	KEEL
Treasury	15	1,049	5	KEEL
Census-House 16L (Census 16L)	16	22,784	3	Delve
Census-House 16H (Census 16H)	16	22,784	3	Delve
Elevators	18	16,559	3	web site <sup>a</sup>
Computer activity (CompAct)	21	8,192	3	web site <sup>a</sup>
Ailerons	40	13,750	3	web site <sup>a</sup>

KEEL knowledge extraction based on evolutionary learning. <http://www.keel.es>, UCI Machine Learning Repository. Collection of regression data sets. <http://archive.ics.uci.edu/ml/datasets.html>, Delve. Regression Data sets. <http://www.cs.utoronto.ca/~delve/data/datasets.html>

<sup>a</sup> L. Torgo, Collection of regression data sets. <http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html>

TIC (85 variables) can be found at KEEL web site. We do not use it because most of the variables are nominal with no order and the output variable is binary. Therefore, this data set is not a regression problem but a classification one.

#### 4.2 Obtained results

The experiments shown in this paper have been performed by a fivefold cross validation with six runs for each data set partition (i.e., a total of 30 runs per problem). Thus, the data set is divided into five subsets of (approximately) equal size. The algorithm is then applied five times to each problem, each problem consisting of 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: 50,000 evaluations, 60 as population size, 0.7 as crossover probability, 0.2 as mutation probability per chromosome. We have not performed any previous analysis to fix these values, so better results may be obtained by tuning them, though we have informally noticed our algorithm is not especially sensitive to any parameter. Strong fuzzy partitions with uniformly distributed triangular MFs are used.

The maximum number of modules is set to four because an SHFS with more modules gets more propagation error and, at the same time, the search space is reduced.

Table 3 collects the obtained results, where  $MSE_{tra}$  and  $MSE_{tst}$  are the approximation error (Eq. 1) values over the training and test data set, respectively;  $\#M$ ,  $\#R$ , and  $\#V$

stand for the number of modules, the number of fuzzy rules, and the number of input variables, respectively;  $\sigma_{\#M}$  is the standard deviation of the number of modules, and  $\bar{x}_{\#R}$  is the average number of variables per rule, which is computed as follows:

$$\bar{x}_{\#R} = \frac{\sum_{i=1}^{\#M} (V_i \times R_i)}{\sum_{i=1}^{\#M} R_i} \quad (3)$$

with  $R_i$  being the number of rules of the  $i$ th module and  $V_i$  the number of variables of the  $i$ th module. This measure indicates the mean complexity of the rules.

Since our algorithm performs multi-objective optimization, several solutions are returned in each run. The average Pareto fronts (30 runs per problem) is computed according to a process based on percentiles. Let  $R$  be the number of runs (30 in our experiments) and  $PS_i$  the Pareto set obtained in the  $i$ th run and ordered by the first objective. Thus, given the average size of the Pareto sets:

$$\bar{p} = \text{round}\left(\frac{\sum_{i=1}^R |PS_i|}{R}\right), \quad (4)$$

the average Pareto set

$$\overline{PS} = \{p_k \in \mathbb{R}^C \mid k \in \{1, \dots, \bar{p}\}\} \quad (5)$$

is built as follows:

$$p_k = (p_{k,1}, \dots, p_{k,C}), p_{k,i} = \frac{\sum_{j=1}^R t_{j,i}^k}{R}, \quad (6)$$

**Table 3** Results obtained with  $MSE_{tra}$  and  $MSE_{tst}$  being the approximation error values over the training and test data set

Method	$MSE_{tra}$	$MSE_{tst}$	$\#M \pm \sigma_{\#M}$	$\#R$	$\bar{x}_{\#R}$	$\#V$	Method	$MSE_{tra}$	$MSE_{tst}$	$\#M \pm \sigma_{\#M}$	$MSE_{tst}$	$\bar{x}_{\#R}$	$\#V$
Laser							Mortgage						
WM	265.20663	278.57597	$1.0 \pm 0.0$	58.4	4.0	4.0	WM	0.25600	0.26806	$1.0 \pm 0.0$	197.2	15.0	15.0
JS	299.46004	323.24366	$6.0 \pm 0.0$	55.6	2.1	4.0	JS	0.90399	0.90393	$7.1 \pm 2.4$	80.6	2.7	5.0
GSHFS w/VS	263.73561	293.05683	$1.2 \pm 0.4$	55.0	3.7	4.0	GSHFS w/VS	0.11376	0.12191	$8.1 \pm 1.2$	101.7	2.9	15.0
GSHFS4M w/VS	263.73561	293.05683	$1.2 \pm 0.4$	55.0	3.7	4.0	GSHFS4M w/VS	0.10971	0.11165	$4.0 \pm 0.0$	131.9	4.2	15.0
VSGA	265.20662	278.57597	$1.0 \pm 0.0$	58.4	4.0	4.0	VSGA	0.11871	0.12598	$1.0 \pm 0.0$	61.5	5.4	5.4
GSHFS	263.73561	293.05683	$1.2 \pm 0.4$	55.0	3.8	4.0	GSHFS	0.10097	0.10848	$2.1 \pm 1.0$	40.7	4.6	7.2
	309.01140	327.12956	$1.3 \pm 0.3$	38.9	2.9	3.3		0.10856	0.11200	$2.0 \pm 0.9$	33.6	3.7	5.5
	342.74012	364.76369	$1.1 \pm 0.2$	25.3	2.3	2.5		0.13511	0.13602	$1.8 \pm 0.9$	27.6	2.9	4.3
	694.76074	728.33412	$1.1 \pm 0.1$	15.0	1.7	1.8		0.19417	0.20913	$1.5 \pm 0.6$	16.5	2.0	2.7
	1,469.78981	1,513.25361	$1.0 \pm 0.0$	5.0	1.0	1.0		0.32981	0.34130	$1.0 \pm 0.0$	5.0	1.0	1.0
ELE2							Treasury						
WM	112,271.49	112,718.84	$1.0 \pm 0.0$	65.0	4.0	4.0	WM	0.80168	0.80909	$1.0 \pm 0.0$	195.2	15.0	15.0
JS	117,172.23	127,342.76	$6.0 \pm 0.0$	36.6	1.3	3.0	JS	0.19888	0.20635	$2.8 \pm 1.6$	18.8	1.8	2.0
GSHFS w/VS	98,205.19	102,993.59	$2.0 \pm 0.0$	41.6	2.8	4.0	GSHFS w/VS	0.11325	0.11924	$10.3 \pm 1.1$	80.0	1.9	15.0
GSHFS4M w/VS	98,205.19	102,993.59	$2.0 \pm 0.0$	41.6	2.8	4.0	GSHFS4M w/VS	0.13011	0.13808	$4.0 \pm 0.0$	130.2	4.4	15.0
VSGA	90,955.89	100,100.11	$1.0 \pm 0$	36.6	3.0	3.0	VSGA	0.16362	0.17451	$1.0 \pm 0.0$	16.8	2.2	2.2
GSHFS	72,231.50	75,695.27	$1.4 \pm 0.5$	15.1	1.9	2.4	GSHFS	0.14692	0.15025	$1.5 \pm 0.8$	17.5	3.0	4.1
	183,426.35	185,362.76	$1.3 \pm 0.2$	12.4	1.7	2.1		0.16304	0.17243	$1.5 \pm 0.7$	16.4	2.5	3.4
	349,446.24	347,696.12	$1.6 \pm 0.4$	10.2	1.3	1.9		0.18550	0.19955	$1.6 \pm 0.9$	14.8	2.0	2.8
	487,699.80	473,206.32	$1.5 \pm 0.3$	7.7	1.1	1.5		0.23274	0.24625	$1.3 \pm 0.5$	10.7	1.6	2.0
	591,402.90	565,461.80	$1.0 \pm 0.0$	5.0	1.0	1.0		0.31949	0.33830	$1.0 \pm 0.0$	5.0	1.0	1.0
DEE							Census 16L						
WM	0.14117	0.22888	$1.0 \pm 0.0$	178.4	6.0	6.0	WM	32,504.7	32,886.8	$1.0 \pm 0.0$	682.0	16.0	16.0
JS	0.23050	0.33550	$9.8 \pm 4.7$	179.6	3.0	6.0	JS	23,814.6	24,107.9	$6.4 \pm 2.3$	62.6	3.1	7.0
GSHFS w/VS	0.13961	0.22728	$1.4 \pm 0.5$	162.5	5.6	6.0	GSHFS w/VS	19,772.1	19,972.3	$8.3 \pm 1.7$	62.0	3.6	16.0
GSHFS4M w/VS	0.13948	0.22713	$1.4 \pm 0.5$	161.3	5.5	6.0	GSHFS4M w/VS	19,796.9	20,069.3	$3.9 \pm 0.1$	114.7	5.1	16.0
VSGA	0.13900	0.22390	$1.0 \pm 0.0$	155.2	5.4	5.4	VSGA	21,506.6	21,804.3	$1.0 \pm 0.0$	79.1	6.4	6.4
GSHFS	0.13728	0.23338	$1.1 \pm 0.2$	155.1	5.6	5.6	GSHFS	17,884.4	18,163.4	$2.6 \pm 1.0$	75.1	5.7	10.5
	0.15308	0.22436	$1.2 \pm 0.3$	87.5	4.0	4.2		18,529.2	18,823.2	$2.5 \pm 1.0$	50.2	4.5	8.2
	0.18253	0.21145	$1.5 \pm 0.5$	38.8	2.7	3.5		20,052.2	20,273.0	$2.6 \pm 0.9$	30.9	3.3	6.6
	0.23023	0.23913	$1.5 \pm 0.6$	14.7	1.9	2.4		23,701.0	23,877.6	$2.2 \pm 1.0$	16.1	2.4	4.3
	0.34664	0.35492	$1.0 \pm 0.0$	4.5	1.0	1.0		41,698.3	41,841.1	$1.0 \pm 0.0$	3.0	1.0	1.0
Concrete							Census 16H						
WM	73.0804	97.0062	$1.0 \pm 0.0$	310.4	8.0	8.0	WM	33,217.9	33,895.9	$1.0 \pm 0.0$	754.6	16.0	16.0
JS	95.5405	113.8403	$13.6 \pm 2.3$	300.2	3.5	8.0	JS	28,557.4	29,022.0	$7.9 \pm 2.0$	88.8	3.2	7.0
GSHFS w/VS	71.7252	88.1597	$1.8 \pm 1.0$	263.4	6.8	8.0	GSHFS w/VS	25,756.6	26,002.0	$8.7 \pm 2.1$	72.0	3.9	16.0
GSHFS4M w/VS	70.3553	88.6661	$1.7 \pm 0.9$	267.7	6.8	8.0	GSHFS4M w/VS	24,395.4	24,758.0	$4.0 \pm 0.6$	133.0	5.5	16.0
VSGA	72.4770	88.9838	$1.0 \pm 0.0$	270.0	7.0	7.0	VSGA	24,905.7	25,318.7	$1.0 \pm 0.0$	104.8	6.7	6.7
GSHFS	50.8595	73.5212	$1.1 \pm 0.2$	312.5	7.0	7.1	GSHFS	22,532.9	22,843.7	$1.8 \pm 0.9$	125.7	7.4	9.6
	55.7862	71.1241	$1.2 \pm 0.5$	231.5	5.4	5.8		23,181.9	23,458.2	$1.8 \pm 0.9$	98.0	5.4	7.6
	70.3104	79.0363	$1.2 \pm 0.4$	133.7	4.1	4.4		24,924.8	25,197.7	$2.3 \pm 1.1$	46.7	3.6	6.2
	112.1841	117.8512	$1.2 \pm 0.4$	48.1	2.6	2.9		27,946.3	28,183.0	$1.9 \pm 0.9$	18.5	2.5	4.0
	240.1951	248.0092	$1.0 \pm 0.0$	3.9	1.0	1.0		50,624.0	50,711.6	$1.0 \pm 0.0$	2.7	1.0	1.0
Census 8L							Elevators						
WM	15,233.8	16,122.2	$1.0 \pm 0.0$	762.8	8.0	8.0	WM	0.33258	0.33485	$1.0 \pm 0.0$	511.0	18.0	18.0
JS	14,503.1	15,187.9	$6.3 \pm 0.7$	86.7	2.9	4.0	JS	0.28862	0.29267	$4.7 \pm 1.0$	35.5	3.0	4.0
GSHFS w/VS	12,750.4	13,090.1	$3.3 \pm 0.6$	145.0	4.2	8.0	GSHFS w/VS	0.23387	0.23466	$10.6 \pm 1.9$	58.8	3.4	18.0
GSHFS4M w/VS	12,546.0	12,973.7	$3.1 \pm 0.6$	166.4	4.6	8.0	GSHFS4M w/VS	0.23721	0.23774	$4.0 \pm 0.0$	105.1	5.2	18.0

**Table 3** continued

Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M ± σ <sub>#M</sub>	#R	$\bar{x}_{\#R}$	#V	Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M ± σ <sub>#M</sub>	MSE <sub>test</sub>	$\bar{x}_{\#R}$	#V
VSGA	13,468.4	14,023.2	1.0 ± 0.0	140.1	5.0	5.0	VSGA	0.24786	0.25033	1.0 ± 0.0	44.6	6.0	6.0
GSHFS	12,739.0	13,210.8	1.6 ± 0.6	169.4	5.6	6.7	GSHFS	0.22158	0.22221	1.7 ± 0.7	21.9	4.3	5.6
	13,431.1	13,875.8	1.6 ± 0.8	98.3	4.3	5.2		0.23308	0.23294	2.1 ± 0.9	16.0	3.0	4.9
	16,045.5	16,516.5	1.6 ± 0.8	39.9	2.9	3.8		0.24437	0.24395	2.0 ± 0.7	11.9	2.4	3.9
	20,584.3	20,950.2	1.8 ± 0.8	19.8	1.8	2.8		0.26514	0.26509	1.6 ± 0.5	7.1	1.8	2.5
	30,995.8	31,407.6	1.0 ± 0.0	5.0	1.0	1.0		0.31726	0.31712	1.0 ± 0.0	2.9	1.0	1.0
Census 8H							Compact						
WM	23,350.4	24,504.3	1.0 ± 0.0	713.4	8.0	8.0	WM	81.7173	82.4732	1.0 ± 0.0	425.6	21.0	21.0
JS	22,037.7	23,707.6	11.2 ± 7.0	291.7	3.1	5.0	JS	44.3714	44.0328	4.3 ± 0.7	21.6	2.5	5.0
GSHFS w/VS	20,734.4	21,846.0	2.7 ± 0.6	419.5	5.6	8.0	GSHFS w/VS	28.0853	28.3043	14.2 ± 1.8	56.0	4.0	21.0
GSHFS4M w/VS	20,648.0	21,717.1	2.4 ± 0.6	444.1	6.0	8.0	GSHFS4M w/VS	35.1213	35.8445	3.9 ± 0.1	6.2	21.0	
VSGA	20,903.4	21,907.5	1.0 ± 0.0		5.4	5.4	VSGA	46.7455	46.5157	1.0 ± 0.0	20.2	5.3	5.3
GSHFS	20,284.0	21,238.4	1.4 ± 0.6	437.5	6.4	6.8	GSHFS	12.6598	12.7747	1.4 ± 0.6	45.5	8.4	9.8
	21,194.7	220,71.9	1.3 ± 0.4	273.7	4.7	5.2		13.1206	13.2273	1.4 ± 0.5	38.9	6.4	7.5
	22,713.9	23,242.2	1.6 ± 0.8	119.3	3.3	4.4		14.7702	14.9839	1.4 ± 0.5	27.7	4.6	5.8
	25,139.6	25,270.0	1.5 ± 0.7	28.6	2.3	2.8		21.2532	21.3664	1.3 ± 0.4	16.8	3.0	3.6
	30,390.6	30,344.6	1.0 ± 0.0	5.0	1.0	1.0		143.9023	144.5856	1.0 ± 0.0	3.0	1.0	1.0
Weank							Ailerons						
WM	9.75605	12.25650	1.0 ± 0.0	456.8	9.0	9.0	WM	0.06452	0.06534	1.0 ± 0.0	1,080.6	40.0	40.0
JS	13.13610	16.56026	6.9 ± 2.4	100.8	3.0	5.0	JS	0.05896	0.05931	6.8 ± 1.6	74.3	3.9	7.0
GSHFS w/VS	7.10852	7.35495	3.3 ± 0.6	148.9	4.8	9.0	GSHFS w/VS	0.06007	0.06049	31.5 ± 3.0	91.3	3.0	40.0
GSHFS4M w/VS	7.10044	7.45815	3.0 ± 0.4	170.0	4.2	9.0	GSHFS4M w/VS	0.04635	0.04695	4.0 ± 0.0	416.1	10.3	40.0
VSGA	7.58657	8.84168	1.0 ± 0.0	140.2	5.4	5.4	VSGA	0.04887	0.04912	1.0 ± 0.0	92.0	8.1	8.1
GSHFS	6.63906	6.84172	1.9 ± 0.8	64.8	3.8	5.7	GSHFS	0.04244	0.04263	2.7 ± 0.9	82.2	6.9	10.6
	7.46367	7.49844	2.0 ± 0.8	40.1	3.0	4.9		0.04777	0.04794	2.5 ± 0.9	60.6	4.9	8.3
	8.63689	8.62772	1.8 ± 0.6	25.7	2.4	3.7		0.05728	0.05760	2.5 ± 0.9	30.3	3.5	6.1
	10.75790	11.00243	1.5 ± 0.5	13.8	1.9	2.6		0.06656	0.06735	2.5 ± 0.8	11.2	2.2	4.4
	13.66512	13.62635	1.0 ± 0.0	2.1	1.0	1.0		0.10148	0.10228	1.0 ± 0.0	2.0	1.0	1.0

#M, #R, and #V stand for the no. of modules, fuzzy rules, and input variables, respectively. σ<sub>#M</sub> is the standard deviation of the no. of modules, and  $\bar{x}_{\#R}$  is the average no. of variables per rule. The results in this table (MSE<sub>tra</sub>/MSE<sub>test</sub>) should be multiplied by 10<sup>5</sup> in the case of Census 8L, Census 8H, Census 16L, and Census 16H

with C being the number of objectives and

$$t_j^k = (t_{j,1}^k, \dots, t_{j,C}^k) = \text{rank} \left( 1 + \frac{(|\text{PS}_i| - 1) \cdot (k - 1)}{\bar{p} - 1}, \text{PS}_i \right) \tag{7}$$

with rank(r, P) being the rth element of the ordered set P. If r is the integer, the element holding this position is directly taken. If not, a linear interpolation is computed. For example, if we are considering two objectives (C = 2) and PS<sub>1</sub> = {(1, 10), (2, 7), (5, 6), (7, 5), (9, 3)}:

$$\begin{aligned} \text{rank}(1, \text{PS}_1) &= (1, 10), \text{rank}(2, \text{PS}_1) = (2, 7), \\ \text{rank}(3, \text{PS}_1) &= (5, 6), \text{rank}(4, \text{PS}_1) = (7, 5), \\ \text{rank}(5, \text{PS}_1) &= 7(9, 3), \text{rank}(3.33, \text{PS}_1) \\ &= (5, 6) \cdot 0.33 + (7, 5) \cdot 0.77 = (6.33, 5.33) \end{aligned}$$

Therefore, in an experiment with three runs (R = 3) where we have the above PS<sub>1</sub>, PS<sub>2</sub> = {(2, 9), (3, 6), (7, 4)}, and

PS<sub>3</sub> = {(2, 10), (4, 8), (6, 5), (8, 3)}, the average Pareto set is:

$$\overline{\text{PS}} = \{(1.67, 9.67), (3.22, 7.22), (5.56, 5.22), (8, 3.33)\}.$$

We show five representative solutions from the average Pareto ordered from low to high MSEs: the first row of each problem is the solution with the highest accuracy, the second row is the first quartile, the third row is the median, the fourth row is the third quartile, and the fifth row is the solution with the lowest accuracy.

The following algorithms have been implemented in order to carry out a study of the algorithm that we propose:

- WM method (Wang and Mendel 1992). Although WM is a simple algorithm and the obtained results are not good, we include it in the analysis because it has been used to learn the rule base, and so, the benefits provided by the hierarchical structure can be observed.

- *JS (Joo and Sudkamp algorithm)*. The algorithm is explained in Appendix 3.
- *GSHFS w/VS (GSHFS without Variable Selection)*. This algorithm is the GSHFS but it does not do variable selection. The crossover operator does module exchanges between SHFS: a different crossing point is selected randomly in each parent; the first offspring inherits from the first parent from the beginning of the chromosome to the crossing point; the rest of the variables are inherited from the second parent but in the same order as the second parent. The second offspring inherits from the second parent from the beginning of the chromosome to the crossing point; the rest of the variables are inherited from the first parent but in relative order of the first parent. Regarding the mutation operator, it uses the exchange, insertion and movement of variables of GSHFS. The algorithm creates SHFS with  $n$  as maximum number of modules, being  $n$  the number of problem's inputs.
- *GSHFS4M w/VS (GSHFS without variable selection creating 4 modules)*. It is the GSHFS w/VS but constrained to a maximum of four modules.
- *VSGA (Variable selection by genetic algorithm)*. It is a variable selection algorithm based on a genetic process. See Appendix 2. It has been implemented because is a fair comparison with our algorithm.

Table 3 shows the results obtained by these algorithms.

The GSHFS w/VS, GSHFS4M w/VS, and VSGA rows show the average of the last most accurate of the obtained Pareto per run which improves the WM solution. The JS rows contain the average of the results obtained by the JS algorithm.

An internal analysis of GSHFS is performed in Sect. 4.4. With this study, we will observe whether our algorithm's benefits are due to the hierarchical structure, the variable selection, or both. Previously, on the next section, we will analyze the obtained results by GSHFS and we will compare it with the results of JS algorithm.

### 4.3 Analysis

Our main objective is to improve the interpretability by means of a precise and compact set of rules, while the accuracy is maintained or improved. When the number of rules of a FS decreases, the system gets better interpretability, but the accuracy is lower and vice versa. This event can be observed clearly in the results obtained in problems with both low and high number of variables. Notice that the fuzzy rule set is learned individually for each module.

We can observe in the problems with a lower number of variables (from four to six variables) that the SHFS

obtained by our algorithm with a higher number of modules has worse accuracy according to the WM method. However, if we observe the problems with a considerable number of variables (more than eight), our algorithm has a better performance. The results obtained by our algorithm in accuracy and complexity are better, the third quartile included, than the results obtained by WM.

In *Dee* problem, we can observe that accuracy is high when the number of rules is high and, consequently, the FS has lower interpretability. According to the WM method, the highest accuracy solution is better in accuracy, having a number of rules and a number of variables lower. In first quartile, the accuracy is slightly increased according to the solution obtained by WM in the  $MSE_{tra}$ , but the  $MSE_{tst}$  is lower and the interpretability decreases by almost half.

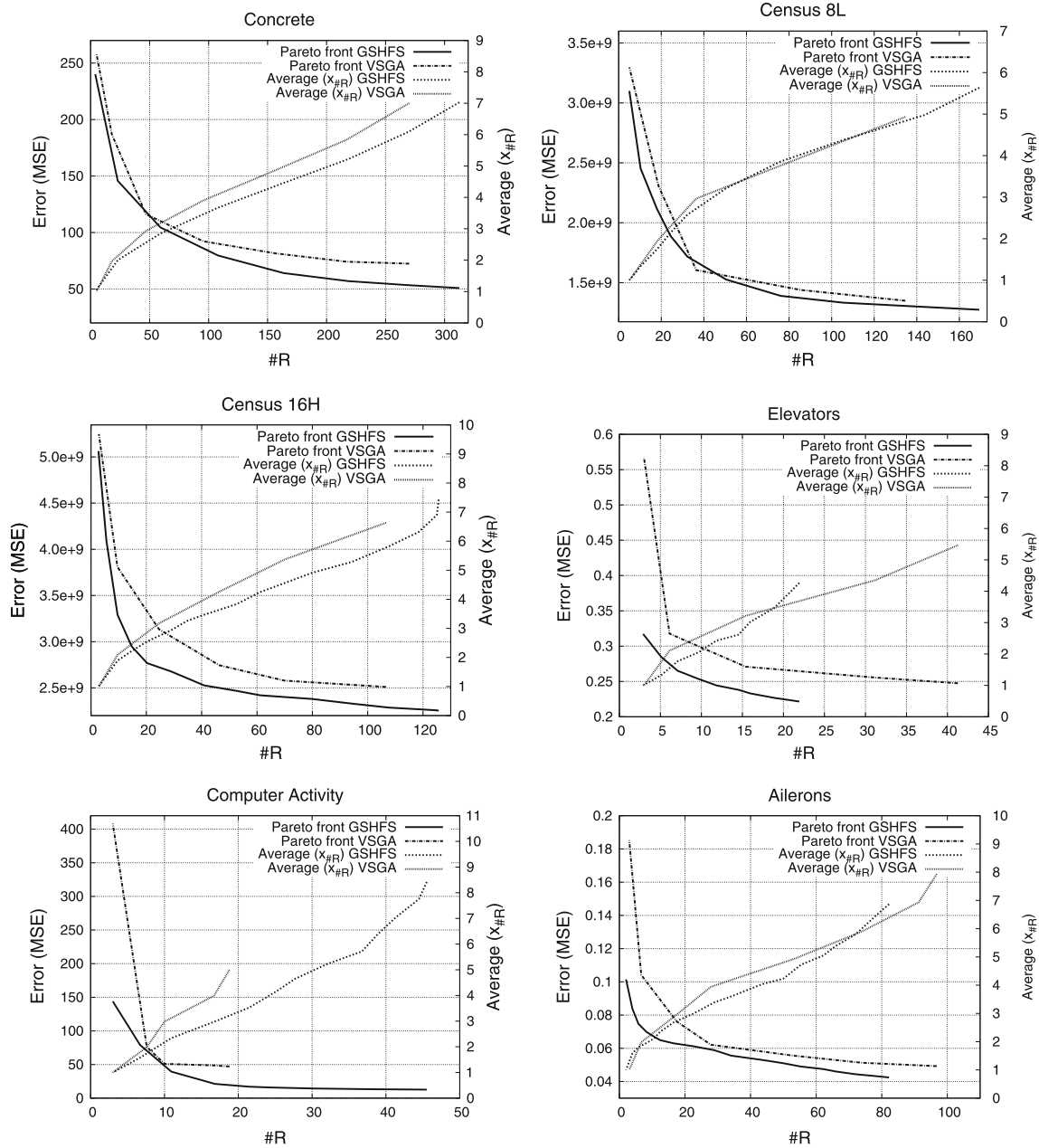
Now, we shall examine the *Elevators* problem, which has 18 variables. The highest accuracy solution obtained by our algorithm is better than the WM solution. In both  $MSE_{tra}$  and  $MSE_{tst}$ , the interpretability is better because the SHFS has a lower number of variables and almost two modules. The number of rules has been decreased by 95 %. The third quartile of the  $MSE_{tra}$  is better in accuracy than WM. The third quartile of the  $MSE_{tst}$  is also more accurate and the number of rules obtained by our algorithm has decreased by 98 %.

In the *Ailerons* problem (40 variables), we can see that our solution with the highest accuracy is better than the solution obtained by WM. We want to emphasize that the number of rules is decreased by 92 % and the number of variables by 75 %. The median of the  $MSE_{tra}$  and the median of the  $MSE_{tst}$  is lower and the interpretability is better than the solution of WM, due to a decrease in the number of variables and its distribution in modules.

Figure 11 shows the average Pareto front obtained by GSHFS in the following large-scale problems: Concrete, Census 8L, Census 16H, Elevators, Computer Activity, and Ailerons. The SHFS achieves higher accuracy when the number of modules tends to be two. If the SHFS has a higher number of modules, the module's carried error will increase considerably. As for complexity, an SHFS has a lower input configuration per module. For this reason, the SHFS' total number of rules is lower and allows major system interpretability.

Figure 12 illustrates some examples of the hierarchical structures obtained by our algorithm. It is an example of non-dominated solutions obtained by GSHFS in a data set partition of *Census16H*, *Elevators*, *Computer Activity*, and *Ailerons*. Its aim is to observe the effect produced by the first module in the SHFS. Therefore, we compare these solutions obtained by GSHFS and by WM to the set of input variables of the last module obtained by GSHFS. We can observe that the  $MSE_{tra}$  and  $MSE_{tst}$  obtained by our algorithm are lower than the values obtained by WM not





**Fig. 11** Average Pareto Front in several problems

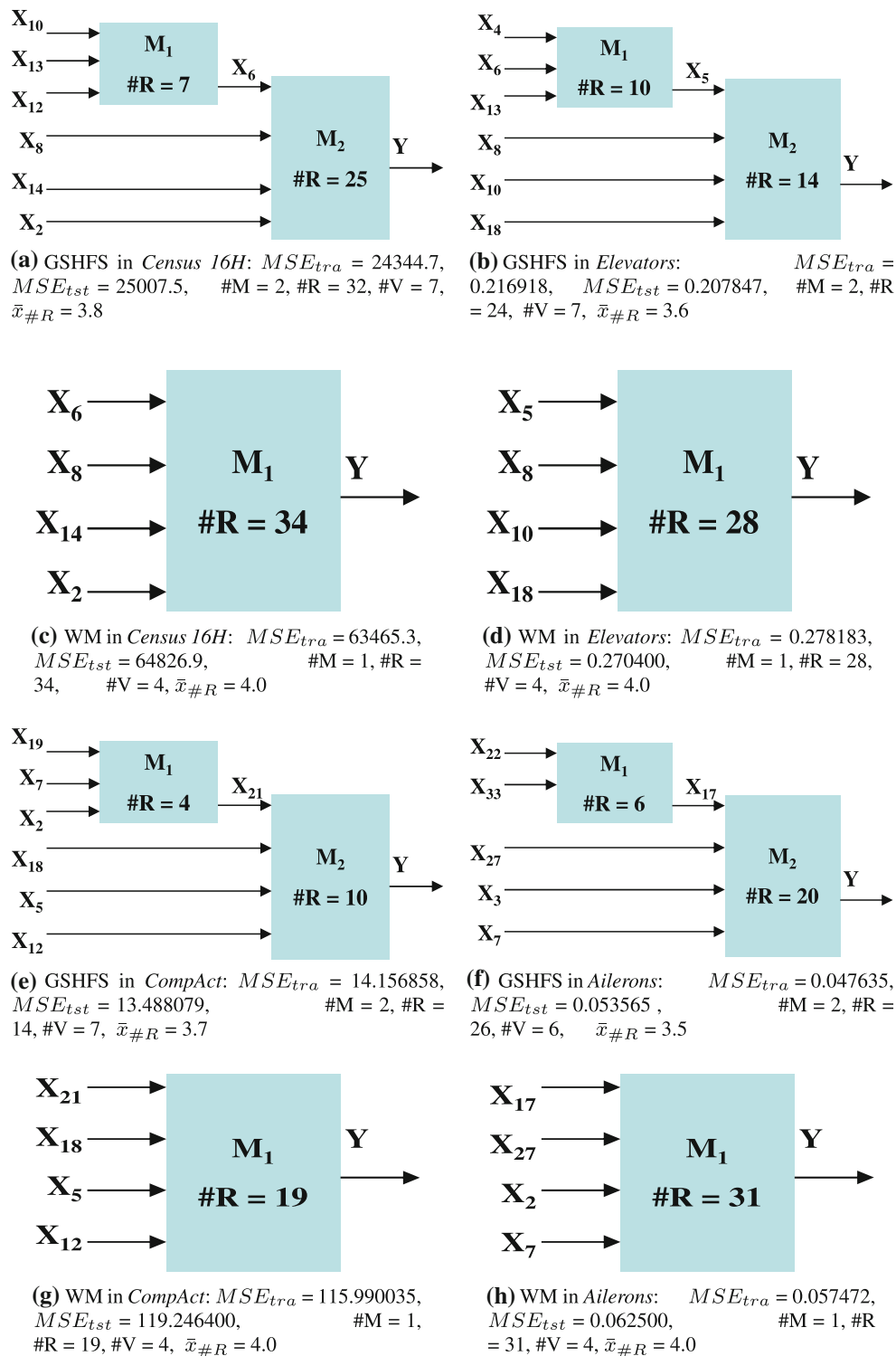
inferencing the endogenous input variable in the last module, decreasing the number of rules by 6 % in *Census16H*, 14 % in *Elevators*, 26 % in *Computer Activity*, and 16 % in *Ailerons*. The average number of variables per rule is lower in the solutions obtained by GSHFS. It allows major system interpretability because the rules are simpler.

The solutions obtained by GSHFS are more promising in accuracy as well as complexity due to it searches in local areas of the search space, and because the rules are simpler. The obtained system is more compact: it has a lower number of variables, which are handed out to different

modules, and so it is easier to interpret the learned rules because they are simpler and there are not many modules (the number of modules is around two).

Table 4 illustrates a real example of a hierarchical rule base. This rule base corresponds to the SHFS in Fig. 12b. The problem consists of 18 inputs. They have been reduced to seven and hierarchically distributed in two modules.

Figure 13 is a bar-chart of the variable dependence rate from the most accurate to the first quartile solutions for all problems. The variable dependence rate is defined as the



**Fig. 12** Example of solutions obtained by GSHFS (a, b, e, f) and the corresponding solutions obtained by WM over the set of input variables of the last module obtained by GSHFS (c, d, g, h). The results in this figure ( $MSE_{tra}/MSE_{tst}$ ) should be multiplied by  $10^5$  in the case of *Census 16H*

number of different variables that depend on other variable(s) at least in one of the most accurate non-dominated solutions (those contained in the first quartile). We consider that variable  $X$  is a dependent variable if it is the output of

one of the modules of the SHFS. The rate has been normalized to a percentage.

The bar-chart shows that there are some complex problems with a high percentage of dependent variables

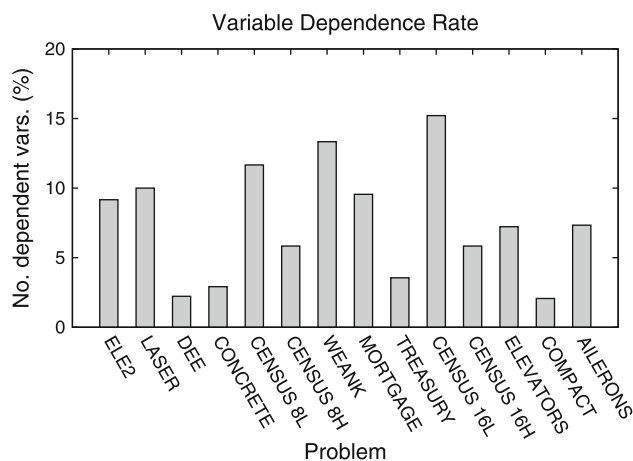
**Table 4** Rule base obtained in elevators problem corresponding to Fig. 12b(a) Rule base of the  $M_1$  module

- R<sub>1</sub>: **IF** q is **and** absRoll is M **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>2</sub>: **IF** q is **and** absRoll is L **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>3</sub>: **IF** q is **and** absRoll is S **and** SaTime4 is S **THEN** curRoll is M  
 R<sub>4</sub>: **IF** q is **and** absRoll is M **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>5</sub>: **IF** q is **and** absRoll is M **and** SaTime4 is M **THEN** curRoll is M  
 R<sub>6</sub>: **IF** q is **and** absRoll is L **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>7</sub>: **IF** q is **and** absRoll is S **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>8</sub>: **IF** q is **and** absRoll is M **and** SaTime4 is S **THEN** curRoll is L  
 R<sub>9</sub>: **IF** q is **and** absRoll is M **and** SaTime4 is M **THEN** curRoll is L  
 R<sub>10</sub>: **IF** q is **and** absRoll is L **and** SaTime4 is S **THEN** curRoll is L

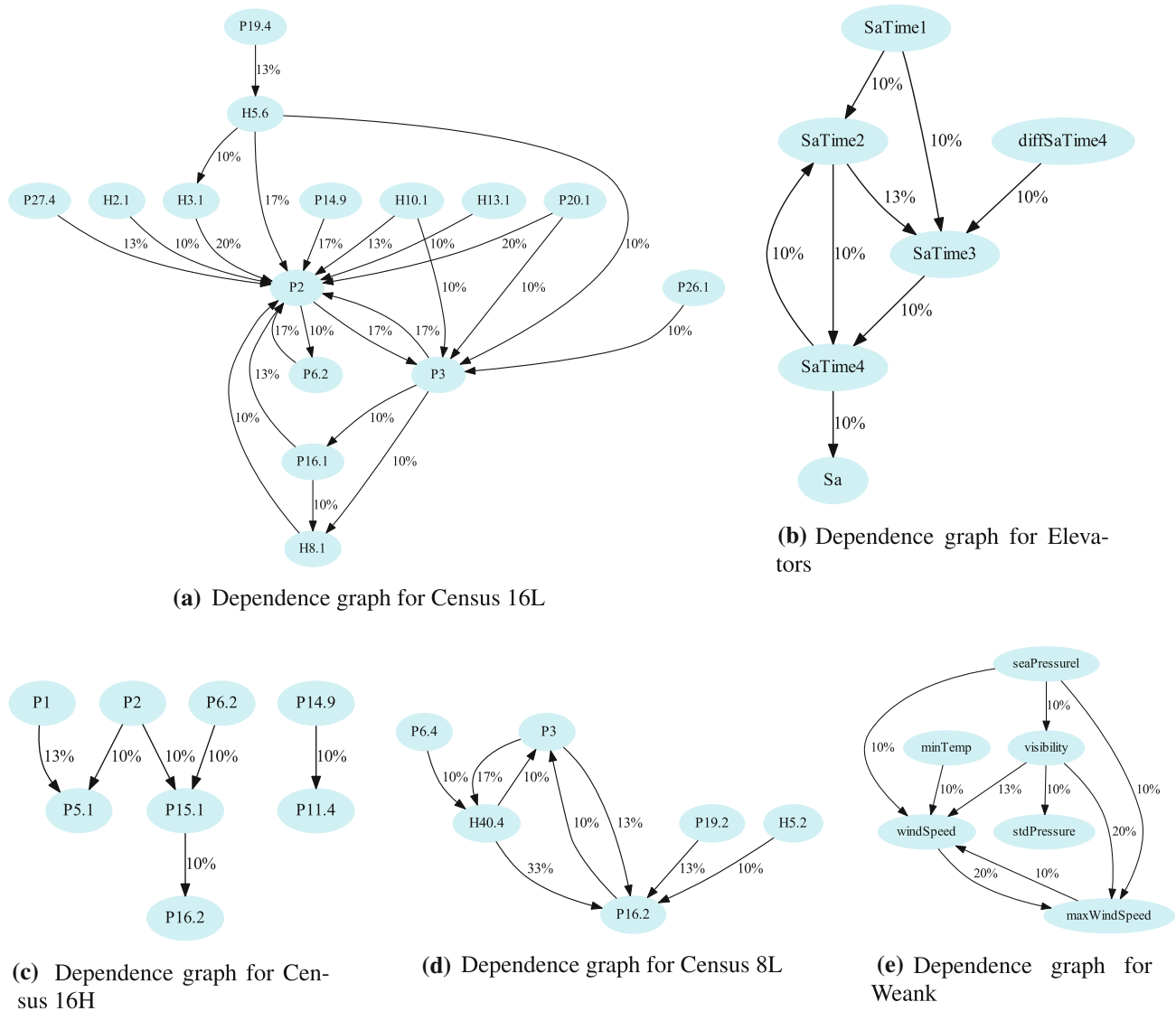
(b) Rule base of the  $M_2$  module

- R<sub>1</sub>: **IF** curRoll is M **and** diffRollRate is M **and** SaTime1 is S **and** Si is L **THEN** Goal is M  
 R<sub>2</sub>: **IF** curRoll is M **and** diffRollRate is M **and** SaTime1 is M **and** Si is L **THEN** Goal is S  
 R<sub>3</sub>: **IF** curRoll is M **and** diffRollRate is M **and** SaTime1 is L **and** Si is L **THEN** Goal is S  
 R<sub>4</sub>: **IF** curRoll is M **and** diffRollRate is L **and** SaTime1 is M **and** Si is L **THEN** Goal is M  
 R<sub>5</sub>: **IF** curRoll is M **and** diffRollRate is L **and** SaTime1 is L **and** Si is L **THEN** Goal is S  
 R<sub>6</sub>: **IF** curRoll is L **and** diffRollRate is S **and** SaTime1 is S **and** Si is M **THEN** Goal is M  
 R<sub>7</sub>: **IF** curRoll is L **and** diffRollRate is S **and** SaTime1 is S **and** Si is L **THEN** Goal is M  
 R<sub>8</sub>: **IF** curRoll is L **and** diffRollRate is S **and** SaTime1 is M **and** Si is L **THEN** Goal is S  
 R<sub>9</sub>: **IF** curRoll is L **and** diffRollRate is M **and** SaTime1 is S **and** Si is M **THEN** Goal is M  
 R<sub>10</sub>: **IF** curRoll is L **and** diffRollRate is M **and** SaTime1 is S **and** Si is L **THEN** Goal is M  
 R<sub>11</sub>: **IF** curRoll is L **and** diffRollRate is M **and** SaTime1 is M **and** Si is L **THEN** Goal is M  
 R<sub>12</sub>: **IF** curRoll is L **and** diffRollRate is M **and** SaTime1 is L **and** Si is L **THEN** Goal is S  
 R<sub>13</sub>: **IF** curRoll is L **and** diffRollRate is L **and** SaTime1 is M **and** Si is L **THEN** Goal is S  
 R<sub>14</sub>: **IF** curRoll is L **and** diffRollRate is L **and** SaTime1 is L **and** Si is L **THEN** Goal is S

(Mortgage, Census 16L, and Ailerons) but no others (Treasury and Computer Activity). This behavior also happens in problems with a low number of variables. For example, Ele2 and Laser have more percentage of dependent variables than Dee or Concrete problems. We can observe that some simple problems (Laser and Ele2) have more percentage of dependent variables than other complex problems (Treasury and Computer Activity). The dependence between variables is not related to the number of variables of the problem. The dependence of variables is linked with the characteristics of the problem. This can be seen clearly on the Census problems. The letters (L or H) denote a very rough approximation to the difficulty of the task. For Low task difficulty, more correlated inputs were chosen as signified by univariate smooth fit of that input on the target. Tasks with high difficulty have had their inputs chosen to make the modeling more difficult due to higher variance or lower correlation of the inputs to the target. Census 16L and Census 16H have the same number of variables and instances, but we can observe in the bar-chart that the variable dependence rate in Census 16L is higher than Census 16H. It means that the variables of Census 16L are more correlated and the learning of dependencies between variables is easier. In this way, an SHFS can work in simple and complex problem.

**Fig. 13** Bar-chart of the variable dependence rate from the most accurate to the first quartile solutions

This interesting behavior deserves a deeper analysis than the one developed here. To do so, a data complexity analysis in a similar manner as made in classification (Ho and Basu 2002; Ho et al. 2006) would be helpful. However, these data complexity measures are not easy to define in regression, so tools as the GSHFS proposed in this paper may help to provide a further insight to understand the



**Fig. 14** Some examples of found dependence graphs (dependencies below 10 % are omitted)

complexity of the problem beyond merely considering its number of variables and instances.

Figure 14 illustrates the correlation between variables by directed graphs in several problems. Each node has been labeled with the real name of the variable and the arcs represent the degree of dependency between pair of variables, where the head is input of a module with the tail as output in the shown percentage of cases. For instance, in Fig. 14a, the variable *H5.6* depends on *P19.4* by 13 % (i.e., in a 13 % of cases the algorithm obtains an accurate solution with a module where *H5.6* is the output and *P19.4* one of its inputs). This graphs may help to understand the correlation among the input variables discovered by GSHFS. As previously stated in Fig. 13, we can observe that the dependencies may differ considerably in problems with the same number of variables and instances as shown in Fig. 14a (Census 16L) and c (Census 16H).

The JS algorithm (Joo and Sudkamp 2009) is one of the most recent approach in this area. For this reason, we thought it would be interesting to implement this approach to compare it with our algorithm.

Table 3 shows the results obtained by JS for the dealt problems. In problems with four variables, the most accurate solution obtained by GSHFS is better than the JS solution in  $MSE_{tra}$  and  $MSE_{tst}$  with a lower number of rules and modules. In the problems with six to nine inputs, the first quartile or the mean of GSHFS obtains better results in  $MSE_{tra}$ ,  $MSE_{tst}$ , number of modules, rules, and variables than the JS approach. In large-scale problems, the third quartile of GSHFS gets better accuracy and a lower number of rules than the JS algorithm.

In *Dee*, the mean value is considerably more accurate than the obtained solution by JS. Regarding the number of modules, GSHFS reduces it by 85 % and the number of

rules by 78 %. In *Census 16L*, the highest accuracy solution obtained by GSHFS improves in accuracy by 25 % and decreases by 59 % the number of modules. The number of rules is a little higher. *Ailerons* is the dealt problem with the highest number of variables. The mean obtained by GSHFS improves the JS solution by 3 % in  $MSE_{tra}$  and  $MSE_{test}$ , the number of rules and modules are decreased by 59 and 63 %, respectively.

GSHFS improves the results obtained by JS and maintain a better trade-off between accuracy and interpretability. All in all, the JS approach obtains a solution with a higher number of modules. In previous sections, we mentioned the carried error increases with a higher number of modules. For this reason, the FS's accuracy is worse. The JS algorithm does not remove variables and so the number of rules is higher. The generated FS is less interpretable due to the use of TSK rules and it needs a complete rule base in order to create the hierarchical fuzzy system. This creates a serious problem when the algorithm considers a higher number of variables, for example, with the *Ailerons* problem.

However, GSHFS solves these problems; it controls the number of modules and variables by means of genetic operators and does not generate new linking variables. In this way, there is a lower number of rules, and as the GSHFS' rules are Mamdani-type, they are more interpretable. For these reasons, it is a very suitable algorithm for large-scale problems.

#### 4.4 Study of the GSHFS' behavior

In this section, we analyze the reasons why our algorithm obtains good results. In order to do this, we compare our algorithm with VSGA and with GSHFS generating one, two, three, and four modules. The aim of this study is to isolate the variable selection benefits and to check if the variable selection or the hierarchical structure has more influence.

The solution with the highest accuracy for each problem obtained by VSGA is shown in Table 3. If we compare the results obtained by VSGA and GSHFS for problems with a number of variables between four and nine, we can observe there is no major difference in accuracy of the results when the problem has four variables (*Laser* and *Ele2*). If our algorithm considers large-scale problems and problems with a number of variables between six and nine, the algorithm's behavior is better. We can observe that the results obtained by GSHFS are better than VSGA, both in accuracy and simplicity. VSGA obtains a FS with a low number of rules, and the inference is not correct.

$\bar{x}_{\#R}$  (Table 3) is a measure which distributes the number of variables, and so, if the number of variables is equal, the

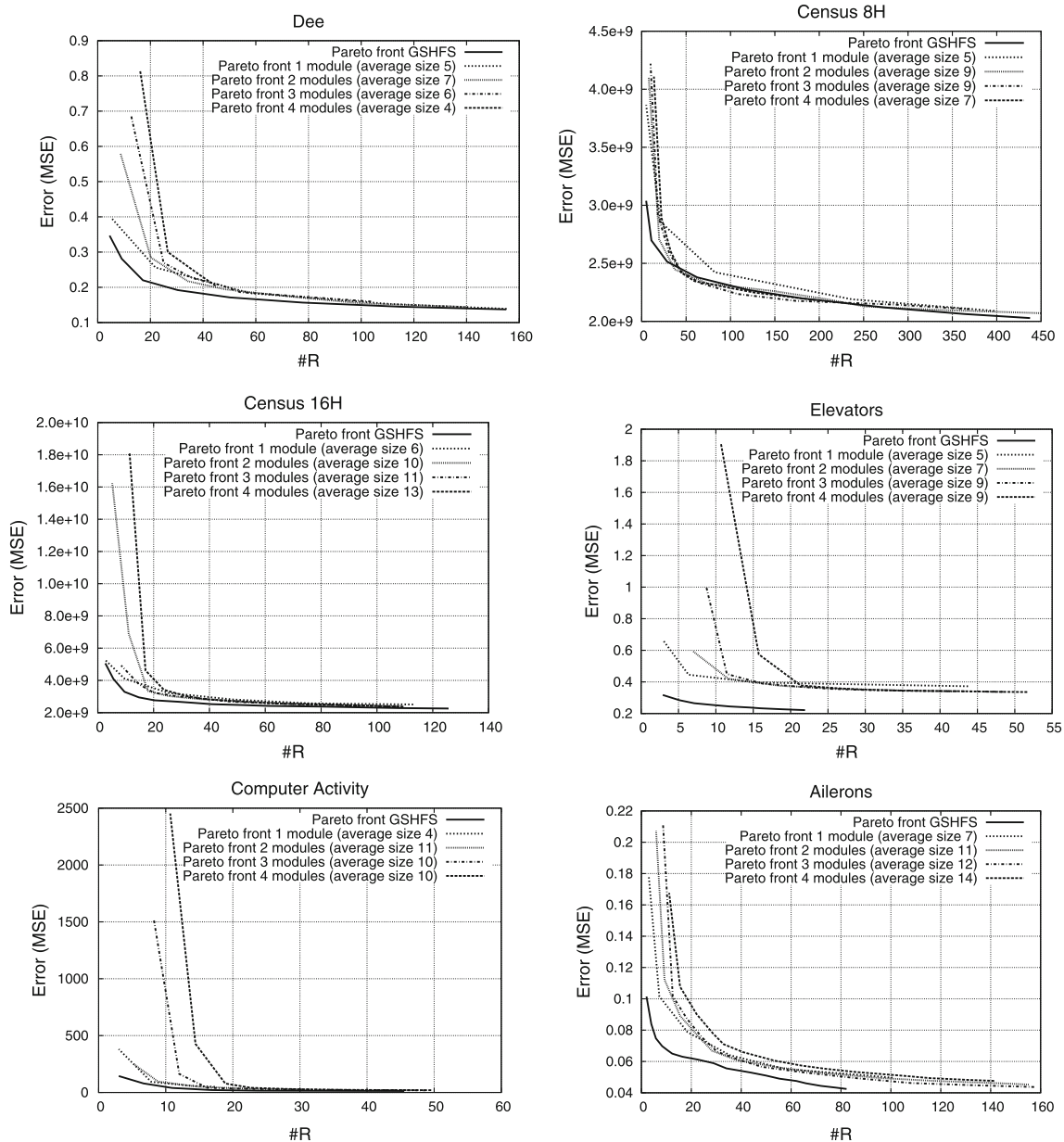
algorithm with a lower  $\bar{x}_{\#R}$  will be better. In this way, we can observe the GSHFS's simplicity; the number of variables per rule in our algorithm is lower than VSGA. This is due to the search space is dealt with a modular way by GSHFS.

Figure 11 depicts the average Pareto Front of GSHFS and VSGA, and the  $\bar{x}_{\#R}$  in *Concrete*, *Census 8L*, *Census 16H*, *Computer Activity*, and *Ailerons*. It shows the GSHFS algorithm is better in accuracy and simplicity (because there is a lower number of variables per rule in each module). If we consider the measure  $\bar{x}_{\#R}$ , the value of this measure in GSHFS is lower along the number of rules than VSGA. We can see this behavior in the rest of the problems, and this indicates to us that the rules generated by GSHFS are simpler and, therefore, more easily interpretable. In *Census 16H*, we can observe clearly how GSHFS is a bit worse in  $\bar{x}_{\#R}$  than VSGA when the number of rules is between 60 and 100 rules more or less.

In order to get further insight into the GSHFS' behavior, we have experimented with other version of our algorithm. We changed the dominance criterion to obtain solutions with one, two, three, and four modules. In this way, we can observe the benefits given by a hierarchical structure. Figure 15 depicts the average Pareto fronts with one, two, three, and four modules obtained in the *Dee*, *Census 8L*, *Census 16H*, *Elevators*, *Computer Activity*, and *Ailerons* problems by our algorithm. In the graph's legend, the average size of each Pareto is shown. These figures illustrate that the combination of this four types of SHFS gets a better accuracy as long as the number of rules is higher because the problem's variables are selected and distributed in the SHFS.

Next, we compare the GSHFS algorithm with GSHFS w/VS and GSHFS4M w/VS. We can observe that GSHFS4M w/VS is more accurate than GSHFS w/VS in some problems, although the number of rules is slightly increased. In large-scale problems, the improvement is more significant than problems with a lower number of variables. In *Concrete*, the  $MSE_{tra}$  is reduced by 2 % and the number of rules is increased by 1.6 % approximately. However, in *Mortgage* the accuracy is improved by 3.6 % although the number of rules is increased by 23 %. We want to notice that GSHFS4M w/VS reduces the WM's  $MSE_{tra}$  and the number of rules by 57 and 33 %, respectively. In this way, we can verify that it is not good to have an SHFS with a higher number of modules, because it will obtain worse accuracy.

If we compare GSHFS with GSHFS w/VS and GSHFS4M w/VS, we will observe that the elimination of variables by means of genetic operators produces improvements in all problems, particularly the large-scale problems. In *Concrete*, the mean improves the solution obtained by



**Fig. 15** Average Pareto front with one, two, three, and four modules in several problems

GSHFS4M w/VS in  $MSE_{tra}$  and number of rules by 0.1 and 50 % respectively.

If we pay attention to *CompAct*, the improvement of the most accurate solution produced by GSHFS is quite considerable; the  $MSE_{tra}$  is reduced by 64 % and the number of rules by 63 %. The third quartile reduces the  $MSE_{tra}$  by 39 %.

#### 4.5 Runtime analysis

To complete the analysis, we would like to show the running time of GSHFS. Table 5 collects the runtime of the GSHFS algorithm, where  $\#VarWM$  indicates the average of

the number of variables per module (and, therefore, per WM method call) during the complete execution, and  $\#Runtime$  is the average and standard deviation runtime. The algorithm was implemented in C language, and all the experiments were executed in a Intel Core 4 Quad, 2.5-GHz CPU, 8 GB of RAM, and Linux Red Hat 5.2.

The algorithm lasts until 12 h in the worst case. It can be a reasonable runtime considering that we are dealing with some large-scale regression datasets (which is not so prevalent in GFSs) and that the proposed algorithm is conceived for off-line learning.

We can find some interesting aspects after a closer look at the table. Indeed, the runtime mainly depends on the

**Table 5** Runtime of the GSHFS algorithm

Data set	#VarWM	Runtime (hh:mm:ss)
Ele2	1.2	00:01:27 ± 00:00:12
Laser	1.6	00:02:03 ± 00:00:27
Dee	2.3	00:01:24 ± 00:00:17
Concrete	3.1	00:08:37 ± 00:02:41
Census-House 8L (Census 8L)	2.4	10:06:29 ± 01:43:53
Census-House 8H (Census 8H)	2.7	10:43:44 ± 01:14:00
Weather-Ankara (WeAnk)	2.1	00:06:19 ± 00:02:11
Mortgage	2.2	00:03:16 ± 00:00:56
Treasury	1.6	00:01:55 ± 00:00:50
Census-House 16L (Census 16L)	2.7	12:32:40 ± 02:50:47
Census-House 16H (Census 16H)	3.1	11:25:17 ± 02:24:37
Elevators	2.1	05:11:55 ± 01:03:27
Computer Activity (CompAct)	3.8	01:26:59 ± 00:27:10
Ailerons	3.6	08:38:05 ± 02:12:17

number of examples of the dataset (see Table 2). This fact would be expected as the WM method generates as many candidate rules as examples exist at the first stage, so it has a clear dependency of this parameter. In this way, problems as Census, Elevators or Ailerons last longer since the number of training examples are between 11,000 and 18,228. Furthermore, it is interesting to note that the number of input variables of the problem is not so relevant to determine the runtime. Thus, given almost the same number of examples, a problem with 8 variables as Concrete lasts longer (about 8 min) than a problem with 15 variable as Treasury (about 2 min). The explanation on this must be found in the real number of rules finally generated. As WM groups the candidate rules by antecedent and then chooses a rule per group to avoid inconsistency, sparse training data may lead to a higher number of rules even when the input dimension and the number of examples is lower. As shown in Table 5, the number of variables used in the modules along the execution of the algorithm directly influences on the runtime and ultimately may represent the difficulty of solving the problem beyond its number of variables. The parameter would be related with more complex features as sparseness of the examples or correlation among the variables.

## 5 Comparison with other GFSs

In order to evaluate the accuracy and interpretability of our proposal in large-scale problems, we compare with some state-of-the-art algorithms in the area of GFSs. Nowadays, most of the GFS algorithms (and particularly in multi-objective optimization) develop an adjustment of the MF parameters in order to reach additional accuracy rates. Therefore, in order to do a fair comparison, we opted by

endowing our proposal with the capability of tuning the fuzzy partitions. However, the main contribution of the paper is not providing an algorithm that generates fuzzy models with excellent accuracy but hierarchical fuzzy models that organize the input variables in a more understandable way without sacrificing much accuracy.

This section is organized as follows: Sect. 5.1 explains the proposed tuning approach of the GSHFS algorithm; Sect. 5.2 describes the experimental setup; Sect. 5.3 shows the obtained results, and Sect. 5.4 analyzes and compares our algorithm with other well-known GFSs.

### 5.1 Proposed tuning algorithm

This approach consists of two genetic operators that move the core of the MFs in order to adapt the fuzzy partitions for an additional accuracy improvement. Strong fuzzy partitions are considered.

1. *Fuzzy partition codification* A double-coding vector scheme to represent the cores of the MFs per variable is used. Figure 16 illustrates it. The interval of the MFs is calculated by means of the parameter (Eq. 8):

$$\varepsilon = \frac{\max - \min}{k - 1} \quad (8)$$

with max and min being the maximum and minimum of the variable interval, respectively,  $k$  the number of labels per variable. In this way, the core of the first label is the *minimum*, the core of the second label is *minimum* +  $\varepsilon$ , the core of the third label is *minimum* +  $2\varepsilon$ , and so on.

2. *Initialization* The first half of the population have been initialized with predefined MFs. The second half was initialized randomly by means of the Algorithm 6.

---

#### Algorithm 6 Random initialization of the MFs

---

```

d = 1/k {k being the number of labels per variable}
for each individual do
  for var = 1 to N do
    {N being the number of variables}
    for l = 1 to labels - 2 do
      r=Choose_random(var, l); {Random number in the label interval}
      Generate_new_core(var,l,r);
    end for
  end for
  for var = 1 to N do
    {N being the number of variables}
    for l = 1 to labels - 2 do
      p=Get_core(var, l);
      p_right=Get_core(var, l+1);
      distance=p_right-p;
      if ( distance < (d/2) ) then
        {The labels are close}
        p_right=p_right+(d/2);
        p=p-(d/2);
        Generate_new_core(var,l,p);
      end if
    end for
  end for
end for

```

---

3. *Crossover operator* This crossover operator is applied per variable to an offspring obtained by means of the GSHFS' crossover. It is embedded into the GSHFS' crossover operator. There are three types of crossing for tuning. It depends on the offspring's common variable selected:

- (a) If the variable inherited by the offspring comes from the two parents, BLX- $\alpha$  crossover will be applied per label. Figure 17 illustrates it.
- (b) If the variable inherited by the offspring comes from one of the two parents, the values of the MFs are copied from the parent into the offspring's inherited variable.

Finally, if the MFs' cores are overlapping each, the MFs vector will be sorted from lowest to highest value.

4. *Mutation operator* The mutation process is as follows:

- (a) Select a random individual  $i$ .
- (b) Choose a random variable  $x$  from  $i$ .
- (c) Choose a random label  $l$  from  $x$ .
- (d) Select a random number  $r$  belonging to  $l$ 's interval.
- (e) Test if  $r$  core are overlapping with the cores on both sides. In the positive case, the new core will be the average of the cores located on both sides of  $r$ .

These genetic operators are used regardless of hierarchical genetic operators. When the 75 % of the evaluations is exceed, the algorithm does not learn hierarchical structures and the MFs tuning is realized.

### 5.2 Experimental setup

The experiments shown in this section have been performed again by a fivefold cross validation with six runs for each data set partition (i.e., a total of 30 runs per problem). Our algorithm has been run with the following parameter values: 100,000 and 300,000 evaluations, 60 as population size, 0.7 as crossover probability, 0.2 as mutation probability per chromosome, and 0.3 as  $\alpha$  BLX crossing parameter. Strong fuzzy partitions with uniformly distributed triangular MFs are used with five fuzzy sets for each linguistic variable. The maximum number of modules is set to four.

The used datasets with 100,000 evaluations are Ele2, Weather Ankara, Mortgage, Treasury, and Computer Activity; while Ele2, Weather Ankara, Mortgage, and Computer Activity have been used for 300,000 evaluations. In this way, we can adapt our experiments to the setup applied by the algorithms considered for comparison.

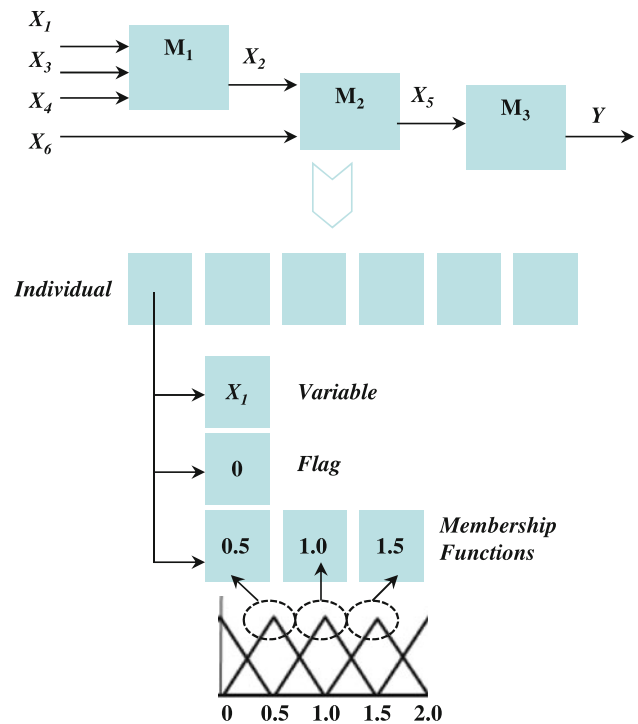


Fig. 16 Example of fuzzy partition coding scheme

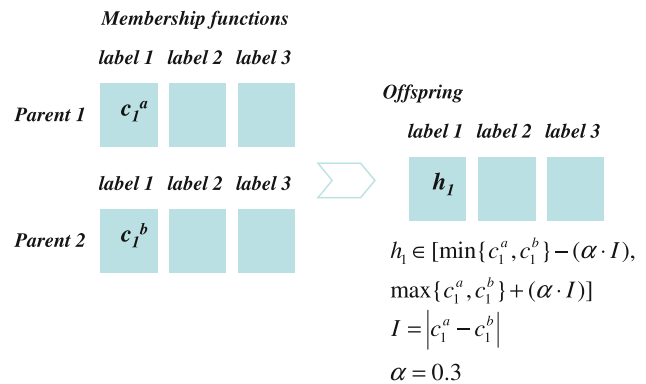


Fig. 17 MFs crossover

### 5.3 Experimental results

Table 6 collects the obtained results, where  $MSE_{tra}$  and  $MSE_{tst}$  are the approximation error values (Eq. 1) over the training and test data set, respectively;  $\#M$ ,  $\#R$ , and  $\#V$  stand for the number of modules, the number of fuzzy rules, and the number of input variables, respectively;  $\sigma_{\#M}$  is the standard deviation of the number of modules, and  $\bar{x}_{\#R}$  is the average number of variables per rule.

The following GFSs have been included for comparison:

- Three single-objective-based methods: GR-MF (Cordón et al. 2001a) learns the granularity for each fuzzy



**Table 6** Results obtained by different algorithms with 100,000 and 300,000 evaluations and five fuzzy sets for each linguistic variable

#Eval	Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M	#R	$\bar{x}_{\#R}$	#V	Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M	#R	$\bar{x}_{\#R}$	#V																													
<b>ELE2</b>																																											
100,000	WM	112,270	112,718	1.0	65.0	4.0	4.0	TREASURY	0.802	0.810	1.0	196.0	15.0	15.0																													
	GR-MF	33,290	37,274	1.0	97.0	4.0	4.0	WM	0.132	0.288	1.0	189.0	15.0	15.0																													
	GA-WM	34,460	37,954	1.0	47.0	4.0	4.0	GR-MF	0.090	0.128	1.0	136.0	15.0	15.0																													
	GLD-WM	22,966	26,768	1.0	33.0	4.0	4.0	GA-WM	0.066	0.090	1.0	70.0	15.0	15.0																													
	GSHFS-tuning	Min	30,933	31,682	1.1	9.9	2.0	2.1	GSHFS-tuning	0.096	0.105	1.3	15.2	2.2	2.8																												
																Q1	130,626	130,553	1.2	8.5	1.7	1.8	Q1	0.106	0.119	1.3	13.4	1.9	2.4														
																PAES-RB	30,908	34,270	1.0	30.0	4.0	4.0	PAES-RB	-	-	-	-	-	-														
																PAES-SF	27,197	31,990	1.0	26.2	4.0	4.0	PAES-SF	-	-	-	-	-	-														
																PAES-SFC	24,464	28,521	1.0	29.1	4.0	4.0	PAES-SFC	-	-	-	-	-	-														
																PAES-SF3	24,543	28,759	1.0	27.0	4.0	4.0	PAES-SF3	-	-	-	-	-	-														
GSHFS-tuning	Min	30,504	31,036	1.0	9.4	1.9	2.0	GSHFS-tuning	-	-	-	-	-	-																													
															Q1	133,398	133,028	1.1	8.1	1.7	1.8	Q1	-	-	-	-	-																
															WM	9,756	12,610	1.0	457.0	9.0	9.0	COMPACT	16,898	24,880	1.0	1539.0	21.0	21.0															
															GR-MF	2,812	14,762	1.0	397.0	9.0	9.0	GR-MF	-	-	-	-	-	-															
300,000	GA-WM	3,044	5,640	1.0	279.0	9.0	9.0	GA-WM	-	-	-	-	-	-																													
															GLD-WM	2,222	4,150	1.0	133.0	9.0	9.0	GLD-WM	-	-	-	-	-																
															GSHFS-tuning	Min	4,567	5,073	1.4	24.5	2.4	3.3	GSHFS-tuning	9,891	10,727	1.7	247.2	7.4	9.1														
																														Q1	6,084	6,424	1.3	16.0	2.1	2.3	Q1	10,641	11,131	1.6	153.2	5.2	6.5
																														PAES-RB	5,400	7,000	1.0	28.3	9.0	9.0	PAES-RB	35,040	37,820	1.0	7.0	21.0	21.0
																														PAES-SF	3,160	3,980	1.0	15.9	9.0	9.0	PAES-SF	12,980	14,320	1.0	6.3	21.0	21.0
PAES-SFC	2,880	3,620	1.0	20.9	9.0	9.0	PAES-SFC	19,080	20,000	1.0	8.4	21.0	21.0																														
PAES-SF3	3,200	3,900	1.0	18.1	9.0	9.0	PAES-SF3	13,140	13,340	1.0	11.6	21.0	21.0																														
GSHFS-tuning	Min	4,390	4,730	1.7	18.9	2.2	3.3	GSHFS-tuning	9,730	10,610	2.0	251.3	7.0	9.4																													
															Q1	6,060	6,330	1.4	14.5	2.0	2.6	Q1	10,310	10,840	1.8	122.6	4.7	6.5															

Table 6 continued

#Eval	Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M	#R	$\bar{x}_{\#R}$	#V	Method	MSE <sub>tra</sub>	MSE <sub>test</sub>	#M	#R	$\bar{x}_{\#R}$	#V	
MORTGAGE															
100,000	WM	0.256	0.268	1.0	199.0	15.0	15.0								
	GR-MF	0.060	0.352	1.0	209.0	15.0	15.0								
	GA-WM	0.040	0.186	1.0	160.0	15.0	15.0								
	GLD-WM	0.032	0.044	1.0	78.0	15.0	15.0								
GSHFS-tuning															
	Min	0.065	0.073	1.5	19.8	2.5	3.3								
	Q1	0.094	0.105	1.3	15.5	2.1	2.6								
300,000	PAES-RB	0.120	0.140	1.0	12.9	15.0	15.0								
	PAES-SF	0.060	0.100	1.0	9.0	15.0	15.0								
	PAES-SFC	0.080	0.160	1.0	10.1	15.0	15.0								
	PAES-SF3	0.060	0.080	1.0	9.0	15.0	15.0								
GSHFS-tuning															
	Min	0.060	0.060	1.4	20.8	2.7	3.3								
	Q1	0.080	0.090	1.3	16.6	2.2	2.8								

GR-MF, GA-WM, and GLD-WM are not available for COMPACT problem. PAES-RB, PAES-SF, PAES-SFC, and PAES-SF3 are not available for TREASURY problem

partition and MF parameters. GA-WM (Cordón et al. 2001b) learns the granularity, scaling factors, and the domains for each variable system. GLD-WM (Alcalá et al. 2007b) learns the knowledge base by obtaining the granularity and the individual lateral displacements of the MFs. The results are extracted from Alcalá et al. (2011a).

- Three versions of the two-objective (2 + 2)M-PAES: PAES-RB learns only rules. PAES-SF and PAES-SFC learn concurrently the RB and the MF parameters by means of the piecewise linear transformation. These algorithms have been taken from Antonelli et al. (2011).
- A three-objective evolutionary algorithm: PAES-SF3 (Antonelli et al. 2011) seeks for different balances among complexity, accuracy, and partition integrity by concurrently learning the RB and MF parameters of the linguistic variables.

Table 6 shows the results of the most accurate solution obtained by our tuning approach (called GSHFS-Tuning), GR-MF, GA-WM, and GLD-WM. These algorithms have been run with 100,000 evaluations and five fuzzy sets for each linguistic variable. This table also presents the FIRST point obtained by PAES-RB, PAES-SF, PAES-SFC, and PAES-SF3. They have been run with 300,000 evaluations and five fuzzy sets for each linguistic variable. Most of datasets in Antonelli et al. (2011) have a lower size. GSHFS-Tuning has been run with 100,000 and 300,000 evaluations in order to compare it with the algorithms previously mentioned. The most accurate *Min* and the one of the first quartile *Q1* solutions (with ascending sorting by error) are shown in this table.

### 5.4 Analysis

In this subsection, we will analyze the ability of hierarchical structures against other GFS algorithms that generates non-hierarchical structures.

We can observe in the problems from 4 to 15 variables that the accuracy as well as the number of rules obtained for GSHFS-Tuning is ranked at intermediate positions against the rest of algorithms. However, in large-scale problems (from 18 to 40 variables) the approach is the most accurate but with a higher number of rules. However, notice that the number of variables per rule is lower in GSHFS-Tuning than the different MOEAs implemented by Antonelli et al. (2011). It means that although the number of rules obtained is high, each individual rule is simpler.

We can observe that with 100,000 evaluations, the GSHFS-Tuning errors are usually higher than GLD-WM but the number of rules is significantly decreased (e.g., by 70 % in *Ele2*).

Compared to PAES-based algorithms with 300,000 evaluations, GSHFS-Tuning's accuracy only improves to PAES-RB algorithm in *Ele2* problem but the simplicity is higher in both the number of rules as in the number of variables per rule, decreased by 65 and 50 %, respectively. If we take a look at the *Computer Activity* problem, we can observe that GSHFS-Tuning obtains the most accurate result. The accuracy is improved by 25 % compared to the result obtained by PAES-SF. GSHFS-Tuning increases the number of rules by 97 %, but we want to notice that the number of variables per rule is lowest and therefore simpler rules are considered. The accuracy is even outperformed by the first quartile solutions and the number of rules is reduced more than a half.

To sum up, the results obtained by GSHFS-Tuning are more accurate in large-scale problems but the number of rules is higher. However, it is difficult to decide which is better because the Pareto fronts of the PAES-based algorithms are not available. Besides, GSHFS-Tuning is compared with algorithms that learn the number of labels, thus reducing the number of rules considerably.

## 6 Conclusion and further work

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

The interpretability is good due to several reasons: (1) the hierarchical structure generates a lower number of variables, (2) the algorithm does not generate artificial linking variables, and so, all of the variables are interpretable because they belong to the system, (3) the rules are simpler because the number of variables per module is lower. At the same time, the accuracy is maintained or improved.

Besides, the hierarchical structures may help to analyze the relationships existing among the input variables thus providing a further insight to understand the complexity of the problem beyond merely considering its number of variables and instances.

The results obtained are promising, which opens a research line in GFSs. As further work, we suggest the study of other mechanisms for a better accuracy (e.g., learning MFs), a detailed study of the interpretability of each rule of the FS, and to extend this algorithm to parallel and hybrid structure learning.

**Acknowledgments** The authors would like to thank Astrophysics Institute of Andalusia (IAA) and Institute for Cross-Disciplinary Physics and Complex Systems (IFISC) [both from the Spanish National Research Council (CSIC)] for providing us with the Grid

nodes used to obtain the experimental results. This work was supported in part by the Spanish Ministry of Science and Innovation (Grant No. TIN2011-28488), the Spanish National Research Council (Grant No. 200450E494), and the Andalusian Government (Grant Nos. P07-TIC-3185 and TIC-2010-6858).

## Appendix 1: Wang–Mendel method

The ad hoc data-driven Mamdani-type fuzzy rule set generation process proposed by Wang and Mendel (1992) is widely known and used because of its simplicity. In our algorithm, GSHFS, it is used in the module's rule base learning. It is based on working with an input–output data pair set representing the behavior of the problem being solved:

$$E = \{e_1, \dots, e_N\}, e_l = (x_1^l, \dots, x_n^l, y_1^l, \dots, y_m^l),$$

with  $N$  being the data set size,  $n$  the number of input variables, and  $m$  the number of output variables. The algorithm consists of the following steps:

1. Consider a fuzzy partition (definitions of the MFs parameters) for each input/output variable.
2. Generate a candidate fuzzy rule set: This set is formed by the rule best covering each example contained in  $E$ . Thus,  $N$  candidate fuzzy rules,  $CR^l$ , are obtained. The structure of each rule is generated by taking a specific example, i.e., an  $(n + m)$ -dimensional real vector, and setting each one of the variables to the linguistic term (associated fuzzy set) best covering every vector component:

$$\begin{aligned} CR^l: & \text{ IF } X_1 \text{ is } A_1^l \text{ and } \dots \text{ and } X_n \text{ is } A_n^l \\ & \text{ THEN } Y_1 \text{ is } B_1^l \text{ and } \dots \text{ and } Y_m \text{ is } B_m^l \\ & A_i^l = \arg \max_{A' \in A_i} \mu_{A'}(x_i^l), B_j^l = \arg \max_{B' \in B_j} \mu_{B'}(y_j^l) \end{aligned}$$

3. Give an importance degree to each candidate rule:
 
$$D(CR^l) = \prod_{i=1}^n \mu_{A_i^l}(x_i^l) \cdot \prod_{j=1}^m \mu_{B_j^l}(y_j^l)$$
4. Obtain a final fuzzy rule set from the candidate fuzzy rule set: To do so, the  $N$  candidate rules are first grouped in  $g$  different groups, each one of them composed of all the candidate rules containing the same antecedent combination. To build the final fuzzy rule set, the rule with the highest degree of importance is chosen in each group. Hence,  $g$  will be both the number of different antecedent combinations in the candidate rule set and the number of rules in the finally generated Mamdani-type fuzzy rule set.

## Appendix 2: Variable selection by genetic algorithm

The VSGA makes a variable selection by means of a multi-objective elitist population-based algorithm in order to

obtain an accurate and interpretable FS. Algorithm 7 shows a pseudocode.

---

#### Algorithm 7 VSGA Algorithm

---

**Require:** Population size, crossover and mutation probability.

Data set:  $D = \{(x, y) | x \in \mathbb{R}^n, y \in \mathbb{R}^m\}$ .

MF definitions.

**Ensure:** Set of non-dominated solutions, each one with a different number of rules/accuracy tradeoff.

Initialization(P);

Evaluation(P);

**while** (*non stop condition*) **do**

  P1  $\leftarrow$  Multiobjective\_Selection(P);

  P2  $\leftarrow$  Crossover(P1);

  P3  $\leftarrow$  Mutation(P2);

  Evaluation(P3);

  P  $\leftarrow$  Multi-objective\_Replacement(P3);

**end while**

---

Each individual is a FS with a module and two genetic operators are applied:

- **Crossover:** It is similar to the GSHFS crossover operator when both parents have a module (see Sect. 3.3.4).
- **Mutation:** This operator chooses probabilistically between to add an unused or remove a used variable to/from an FS. This is shown in Algorithm 8.

The replacement strategy used is NSGA-II Deb et al. (2002) algorithm, the inference mechanism considered is the Max–Min inference scheme, and two objectives are considered: the MSE and the maximum number of rules. The FS' Mamdani-type fuzzy rule set is learned by WM method Wang and Mendel (1992).

---

#### Algorithm 8 VSGA Mutation

---

**Require:** FS Fuzzy System.

$r = U[0,1]$ ; {Uniform probability}

**if** ( $r < 0.5$  and number of unused variables  $> 0$ ) **then**

  Insert\_random\_unused\_variable(FS);

**else**

**if** (number of used variables  $> 1$ ) **then**

    Remove\_random\_used\_variable(FS);

**end if**

**end if**

---

### Appendix 3: Joo and Sudkamp's approach

The Joo and Sudkamp's (2009) algorithm consists of the conversion from a single-layer FS into a two-layer HFS. The first step is to establish what is the number of variables belonging to each layer is, as follows: let  $n$  be the number of variables and  $k$  the number of labels, with  $k \geq 2$ ; the number of variables in the first layer is calculated by the equation:

$$p^* = \frac{n - \log_k 2}{3} \quad (9)$$

The number of variables which minimize the number of rules is calculated in this way:

$$R(p)_{\min} = \min \{R(\lfloor p^* \rfloor), R(\lfloor p^* \rfloor + 1)\} \quad (10)$$

where  $\lfloor x \rfloor$  is the greatest integer less than or equal to  $x$  and  $R(x)$  is obtained as follows:

$$R(x) = k^x k^x + k^{n-x}. \quad (11)$$

The next step is to create the first layer's modules: the complete rule base is generated and the rules are put into groups according to the variables' antecedent belonging to the second layer with the same label value. Each group makes up a module in the first layer. If there are  $p$  variables in the first label, the algorithm will generate  $k^{n-p}$  modules and each module will have  $k^p$  rules. The algorithm includes a hierarchical rule reduction mechanism which eliminates the first layer's modules with lineal dependence by means of the calculation of the matrix rank called RB. Each  $i$  row of the RB matrix represents a module and each column is the consequent of the rule  $j$  of the module  $i$ . The number of modules in the first layer is obtained by the rank of the RB matrix. It uses singleton fuzzification, product inference, and integer average defuzzification.

Next, the second layer is created. Unlike standard FSs, the second layer FSs receives both the values of the rest input variables and the values of the first layer's rule consequents as inputs. The algorithm generates the complete TSK-type rule base with  $n - p$  variables. The rules' consequents are constructed by means of the combination of the inferred values of the first layer's modules with real constants. These constants are obtained from a set of dot-matrix operations with the RB matrix.

Our JS implementation starts from a reduced set of variables by VSGA per run, instead of generating the complete rule base. The objective of this is to obtain a comparable JS algorithm in the same conditions that the rest of implemented algorithms. The main problem is the RB matrix construction, because some matrix cells have no associated value due to the lack of the rules. We decided to fill the cells with the intermediate value of the output variable in order to complete the empty cells of the matrix and then, insert the associated rule into the filled cell in the corresponding module. This insertion does not produce major changes in the first layer's output due to our use of the FITA (First Inference, Then Aggregate) inference and we can consider it as a simplified TSK.

### References

- Aja-Fernández S, Alberola-López C (2008) Matriz modeling of hierarchical fuzzy systems. *IEEE Trans Fuzzy Syst* 16(3):585–599

- Alcalá R, Alcalá-Fdez J, Casillas J, Cordon O, Herrera F (2006) Hybrid learning models to get the interpretability-accuracy trade-off in fuzzy modeling. *Soft Comput* 10:717–734
- Alcalá R, Alcalá-Fdez J, Gacto M, Herrera F (2007a) Rule base reduction and genetic tuning of fuzzy systems based on the linguistic 3-tuples representation. *Soft Comput* 11:401–419
- Alcalá R, Alcalá-Fdez J, Herrera F, Otero J (2007b) Genetic learning of accurate and compact fuzzy rule based systems based on the 2-tuples linguistic representation. *Int J Approx Reason* 44(1): 46–64
- Alcalá R, Gacto M, Herrera F (2011a) A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems. *IEEE Trans Fuzzy Syst* 19:666–681
- Alcalá R, Nojima Y, Herrera F, Ishibuchi H (2011b) Multiobjective genetic fuzzy rule selection of single granularity-based fuzzy classification rules and its interaction with the lateral tuning of membership functions. *Soft Comput* 15(12):2303–2318
- Antonelli M, Ducange P, Lazzerini B, Marcelloni F (2011) Learning knowledge bases of multi-objective evolutionary fuzzy systems by simultaneously optimizing accuracy, complexity, and partition integrity. *Soft Comput* 15(12):2335–2354
- Benítez A, Casillas J (2009) Genetic learning of serial hierarchical fuzzy systems for large-scale problems. In: Proceedings of Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference (IFSA-EUSFLAT 2009). Lisbon, pp 1751–1756
- Casillas J, Carse B (2009) Genetic fuzzy systems: recent developments and future directions. *Soft Comput* 13:417–418 (special issue)
- Casillas J, Cordon O, del Jesús M, Herrera F (2005) Genetic tuning of fuzzy rule deep structures preserving interpretability and its interaction with fuzzy rule set reduction. *IEEE Trans Fuzzy Syst* 13(1):13–29
- Chen Y, Dong J, Yang B (2004) Automatic design of hierarchical ts-fs model using ant programming and pso algorithm. In: Bussler C, Fensel D (eds) Proceedings 12th international conference on artificial intelligence, methodology, systems and applications. Lecture notes on artificial intelligence, LNAI 3192, pp 285–294
- Chen Y, Yang B, Abraham A, Peng L (2007) Automatic design of hierarchical Takagi-Sugeno type fuzzy systems using evolutionary algorithms. *IEEE Trans Fuzzy Syst* 15(3):385–397
- Cheong F (2007) A hierarchical fuzzy system with high input dimensions for forecasting foreign exchange rates. In: IEEE Congress on Evolutionary Computation, CEC, pp 1642–1647
- Chiu S (1996) Selecting input variables for fuzzy models. *J Intell Fuzzy Syst* 4(4):243–256
- Cordon O, Herrera F, Villar P (2001a) Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Trans Fuzzy Syst* 9(4):667–674
- Cordon O, Herrera F, Magdalena L, Villar P (2001b) A genetic learning process for the scaling factors, granularity and contexts of the fuzzy rule-based system data base. *Inf Sci* 136:85–107
- Deb K, Pratap A, Agarwal S, Meyarevian T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans Evol Comput* 6(2):182–197
- Duan J, Chung F (2002) Multilevel fuzzy relational systems: structure and identification. *Soft Comput* 6(2):71–86
- Gacto M, Alcalá R, Herrera F (2009) Adaptation and application of multi-objective evolutionary algorithms for rule reduction and parameter tuning of fuzzy rule-based systems. *Soft Comput* 13:419–436
- Gaweda A, Scherer R (2004) Fuzzy number-based hierarchical fuzzy system, vol 3070. Lecture notes in computer sciences. Springer, Berlin, pp 302–307
- González A, Pérez R (2001) Selection of relevant features in a fuzzy genetic learning algorithm. *IEEE Trans Syst Man Cybern Part B Cybern* 31(3):417–425
- Ho TK, Basu M (2002) Complexity measures of supervised classification problems. *IEEE Trans Pattern Anal Mach Intell* 24(3):289–300
- Ho TK, Basu M, Law M (2006) Measures of geometrical complexity in classification problems. In: Data complexity in pattern recognition. Springer, Berlin, pp 1–23
- Holve R (1998) Investigation of automatic rule generation for hierarchical fuzzy systems. In: Fuzzy systems proceedings, vol 2. IEEE World Congress on Computational Intelligence, pp 973–978
- Hong T, Chen J (1999) Finding relevant attributes and membership functions. *Fuzzy Sets Syst* 103(3):389–404
- Hong X, Harris C (2001) Variable selection algorithm for the construction of mimo operating point dependent neurofuzzy networks. *IEEE Trans Fuzzy Syst* 9(1):88–101
- Ishibuchi H, Nozaki K, Yamamoto N, Tanaka H (1995) Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Trans Fuzzy Syst* 3(3):260–270
- Jelleli T, Alimi A (2005) Improved hierarchical fuzzy control scheme. *Adapt Natural Comput* 1:128–131
- Jelleli T, Alimi A (2010) Automatic design of a least complicated hierarchical fuzzy system. In: 6th IEEE World Congress on computational intelligence, pp 1–7
- Jin Y (2000) Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement. *IEEE Trans Fuzzy Syst* 8(2):212–221
- Joo M, Lee J (1999) Hierarchical fuzzy control scheme using structured Takagi-Sugeno type fuzzy inference. Proceedings of IEEE international fuzzy systems conference, Seoul, In, pp 78–83
- Joo M, Lee J (2002) Universal approximation by hierarchical fuzzy system with constrains on the fuzzy rule. *Fuzzy Sets Syst* 130(2): 175–188
- Joo M, Sudkamp T (2009) A method of converting a fuzzy system to a two-layered hierarchical fuzzy system and its run-time efficiency. *IEEE Trans Fuzzy Syst* 17(1):93–103
- Lee H, Chen C, Chen J, Jou Y (2001) An efficient fuzzy classifier with feature selection based on fuzzy entropy. *IEEE Trans Syst Man Cybern Part B Cybern* 31(3):426–432
- Lee M, Chung H, Yu F (2003) Modeling of hierarchical fuzzy systems. *Fuzzy Sets Syst* 138(2):343–361
- Maeda H (1996) An investigation on the spread of fuzziness in multi-fold multi-stage approximate reasoning by pictorial representation under sup-min composition and triangular type membership function. *Fuzzy Sets Syst* 80(2):133–148
- Nojima Y, Alcalá R, Ishibuchi H, Herrera F (2011) Special issue on evolutionary fuzzy systems. *Soft Comput* 15(12):2299–2301
- Raju G, Zhou J, Kisner R (1991) Hierarchical fuzzy control. *Int J Control* 54(5):1201–1216
- Salgado P (2008) Rule generation for hierarchical collaborative fuzzy system. *Appl Math Modell Sci Direct* 32(7):1159–1178
- Shimozima K, Fukuda T, Hasegawa Y (1995) Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *Fuzzy Sets Syst* 71(3): 295–309
- Tan F, Fu X, Zhang Y, Bourgeois A (2008) A genetic algorithm-based method for feature subset selection. *Soft Comput* 12: 111–120
- Taniguchi T, Tanaka K, Ohtake H, Wang H (2001) Model construction, rule reduction, and robust compensation for generalized form of Takagi-Sugeno fuzzy systems. *IEEE Trans Fuzzy Syst* 9(4):525–538

- Torra V (2002) A review of the construction of hierarchical fuzzy systems. *Int J Intell Syst* 17(5):531–543
- Wang L (1998) Universal approximation by hierarchical fuzzy systems. *Fuzzy Sets Syst* 93(2):223–230
- Wang L (1999) Analysis and design of hierarchical fuzzy systems. *IEEE Trans Fuzzy Syst* 7(5):617–624
- Wang L, Mendel J (1992) Generating fuzzy rules by learning from examples. *IEEE Trans Syst Man Cybern* 22(6):1414–1427
- Wang D, Zeng X, Keane J (2006) Learning for hierarchical fuzzy systems based on gradient-descent method. In: *Proceedings of IEEE international conference on fuzzy systems*, pp 92–99
- Xiong N, Funk P (2006) Construction of fuzzy knowledge bases incorporating feature selection. *Soft Comput* 10:796–804
- Zajaczkowski J, Verma B (2012) Selection and impact of different topologies in multilayered hierarchical fuzzy systems. *Appl Intell* 36(3):564–584
- Zeng X, Goulermas J, Liatsis P, Wang D, Keane J (2008) Hierarchical fuzzy systems for function approximation on discrete input spaces with application. *IEEE Trans Fuzzy Syst* 16(5):1197–1215
- Zhang X, Zhang N (2006) Universal approximation of binary-tree hierarchical fuzzy system with typical FLUs. *Lecture notes in computer science*, vol 4114. Springer, Berlin, pp 177–182