Evol. Intel. (2009) 2:5–19
DOI 10.1007/s12065-009-0025-0

SPECIAL ISSUE

Evolutionary parallel and gradually distributed lateral tuning of fuzzy rule-based systems

I. Robles · R. Alcalá · J. M. Benítez · F. Herrera

Received: 16 June 2009 / Revised: 24 September 2009 / Accepted: 25 September 2009 / Published online: 22 October 2009

Abstract The tuning of Fuzzy Rule-Based Systems is often applied to improve their performance as a post-processing stage once an initial set of fuzzy rules has been extracted. This optimization problem can become a hard one when the size of the considered system in terms of the number of variables, rules and, particularly, data samples is big. Distributed Genetic Algorithms are excellent optimization algorithms which exploit the nowadays available parallel hardware (multicore microprocessors and clusters) and could help to alleviate this growth in complexity. In this work, we present a study on the use of the Distributed Genetic Algorithms for the tuning of Fuzzy Rule-Based Systems. To this end, we analyze the application of a specific Gradual Distributed Real-Coded Genetic Algorithm which employs eight subpopulations in a hypercube topology and local parallelization at each subpopulation. We tested our approach on nine real-world datasets of different sizes and with different numbers of variables. The empirical performance in solution quality and computing time is assessed by comparing its results with those from a highly effective sequential tuning algorithm. The results show that the distributed approach achieves better results in terms of quality and execution time as the complexity of the problem grows.

I. Robles (✉) · R. Alcalá · J. M. Benítez · F. Herrera
Dept. of Computer Sciences and Artificial Intelligence, University of Granada, 18071 Granada, Spain
e-mail: ignaciorobles@gmail.com

R. Alcalá
e-mail: alcala@decsai.ugr.es

J. M. Benítez
e-mail: J.M.Benitez@decsai.ugr.es

F. Herrera
e-mail: herrera@decsai.ugr.es

Keywords Genetic fuzzy system · Fuzzy rule-based systems · Distributed genetic algorithms · Genetic tuning

1 Introduction

Fuzzy rule-based systems (FRBS) have become a wide choice when addressing modeling and system identification problems [1–4]. One of the most popular approaches for the design of FRBSs is the hybridization between fuzzy logic [5, 6] and Genetic Algorithms (GAs) [7, 8] leading to the well-known Genetic Fuzzy Systems (GFSs) [9–11]. A GFS is basically a fuzzy system augmented by a learning process based on evolutionary computation, which includes GAs, genetic programming, and evolutionary strategies, among other evolutionary algorithms [12].

The predominant type of GFS is that focused on FRBSs, since the automatic definition of FRBSs can be seen as an optimization or search problem, and GAs are a well known and widely used global search technique with the ability to explore a large search space for suitable solutions only requiring a performance measure. In addition to their ability to find near optimal solutions in complex search spaces, the generic code structure and independent performance features of GAs make them suitable candidates to incorporate a priori knowledge. In the case of FRBSs, this a priori knowledge may be in the form of linguistic variables [13], fuzzy membership function (MF) parameters, fuzzy rules, number of rules, etc. These capabilities extended the use of GAs in the development of a wide range of approaches for designing FRBSs over the last few years.

In this framework, a widely-used technique to enhance the performance of FRBSs is the genetic tuning of MFs [14–19]. It consists of improving a previous definition of

the Data Base (DB) once the Rule Base (RB) has been obtained. The classic approaches to perform genetic tuning [18, 19] consist of using a GA in order to refine the definition parameters that identify the MFs associated to the linguistic terms comprising the initial DB.

Since the real aim of the genetic tuning process is to find the best global configuration of the MFs and not only to find independently specific ones, this optimization problem can become a hard one when the size of the considered system in terms of the number of variables, rules and, particularly, data samples (typically used to guide the search) is big. Moreover, the computing time consumed by these approaches grows with the complexity of the search space.

In order to deal with this complexity, Distributed Genetic Algorithms (DGAs) [20–22] are found to be excellent optimization algorithms for high dimensional problems. They are able to take advantage of the parallel hardware and software that has become very affordable and broadly available nowadays. Clear examples in this line are multicore processors and linux clusters [23–25]. This situation makes them perfect to deal with complex search spaces.

In this work, we present a study on the use of the Distributed Genetic Algorithms for the tuning of FRBS from two points of view: solution quality and computing time improvements. To this end, we analyze the application of a specific Gradually Distributed Real-Coded Genetic Algorithm (GDRCGA) to perform an effective genetic tuning of FRBSs [26]. This algorithm employs eight subpopulations in a hypercube topology [27], including local parallelization at each subpopulation, and makes use of a particular linguistic rule representation model that was proposed in [15] to perform a genetic *lateral tuning of MFs*. This approach is based on the linguistic 2-tuples representation [28] which simplifies the search space by considering only one parameter per MF and, therefore, eases the derivation of optimal models, particularly in complex or high-dimensional problems.

We tested our approach on nine real-world problems with a number of variables ranging from 2 to 21 and a number of samples ranging from 495 to 8192. The empirical performance in solution quality and computing time has been assessed by comparing the results of the distributed approach with those obtained from the specialized sequential algorithm, proposed in [15], to perform a lateral tuning of the MFs. To assess the results obtained by both algorithms, we have applied a nonparametric statistical test [29–32] for pair-wise comparisons. The results show that the distributed approach achieves better results in terms of quality and execution time as the complexity of the problem grows.

This paper is structured as follows. In Sect. 2, DGAs are presented and briefly discussed. In Sect. 3, the lateral tuning of FRBSs problem is stated and an efficient sequential specialized algorithm is reviewed. Section 4 describes the DGA used for FRBS tuning and explains how the local parallelization is performed at each subpopulation. An empirical evaluation of the distributed algorithm on nine datasets is shown and discussed in Sect. 5. Some conclusions and final remarks are given in Sect. 6. Finally, Appendix describes the Wilcoxon signed-rank test used for the experimental analysis.

2 Preliminaries: basic concepts about DGAs

The availability of extremely fast and low cost parallel hardware in the last few years benefits the investigation on new approaches to existing optimization algorithms. The key of these new approaches is achieving gains not only in time, which is somehow inherent to parallel computation, but also gains in quality of the solutions found.

Generally, there are two ways to parallelize GAs. The first way is by means of local parallelization: fitness evaluation of the individuals and, sometimes, the application of the genetic operators are carried out in a parallel way [33, 34]. The second way is by means of global parallelization: complete subpopulations evolve in parallel [27, 35–41] and these algorithms are known as distributed genetic algorithms (DGAs). While the first one is only achieving gains in time, the second one is also able to improve the global performance of the underlying algorithm, subsequently achieving additional gains in the quality of the final solutions. In fact, DGAs [20, 21] are excellent optimization algorithms and have proven to be an interesting approach when trying to cope with large scale problems and specifically when the classic approaches take too long to give a proper solution.

In this section, our goal is to present an introductory vision of the distributed models. Section 2.1 presents a taxonomy of the state-of-the-art of DGAs. In Sect. 2.2, the key elements to obtain a well-designed DGA are presented.

2.1 Taxonomy of distributed genetic algorithms

Several categorizations of DGAs can be found in the literature [20–22] according to a wide range of criteria. Some of the most used categories when referring to DGAs are:

- According to the migration policy:
  - Isolated: no migrations between subpopulations. These DGAs are also known as Partitioned Genetic Algorithms [41].

  - Synchronous: migrations between subpopulations are synchronized, for example, they are carried out at the same time [38, 41].
  - Asynchronous: migrations are carried out when some events occur, generally related to the activity of subpopulations [37].
- According to the connection schema:
  - Static schema: connections between subpopulations are established at the start of the execution and they are not modified.
  - Dynamic schema: connection topology changes dynamically along the execution of the algorithm. Connection reconfigurations may occur depending on the degree of evolution of the subpopulations.
- According to the homogeneity:
  - Homogeneous: genetic operators are the same for all subpopulations as well as parameters, fitness function, coding scheme, etc. The vast majority of DGAs proposed in the literature are homogeneous.
  - Heterogeneous: subpopulations are all alike [36, 39, 40]. They can differ from the parameters used, genetic operators, coding scheme, etc. One example of these heterogeneous GAs are the Gradually Distributed Genetic Algorithms where genetic operators are applied with different intensities [27].
- According to the granularity:
  - Coarse-grained parallelization: The population is split into small subpopulations that are assigned to different processors. Each subpopulation evolves independently and simultaneously according to a GA. Periodically, a migration operator exchanges individuals among subpopulations, which gives them some additional diversity.
  - Fine-grained parallelization: The population is split into a big number of small subpopulations. Generally only one subpopulation is assigned to each processor. The selection and crossover operators are applied considering adjacent individuals. For example, each individual chooses its best neighbor for crossover (Fig. 1) and the resulting individual replaces the original one. When a single individual is assigned to each processor, this type of algorithms are known as Cellular Genetic Algorithms [35].

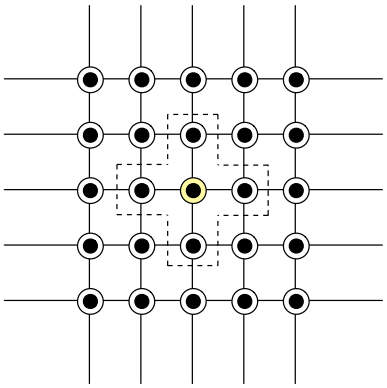

Fig. 1 Cellular genetic algorithm: a extreme case of fine-grained parallelization

2.2 Design of distributed genetic algorithms

There are two classic problems [27] in DGAs. A main drawback in DGAs is that the insertion of a new individual coming from a different subpopulation may not be effective. The new individual could be highly incompatible with the receiving subpopulation and therefore it might be ignored or conquer the subpopulation. This probably happens when subpopulations involved are at different stages of evolution.

The arrival of a highly evolved individual coming from a *strong* subpopulation will result in a higher selection ratio than for local individuals which are less evolved. In this way, the subpopulation that sends the highly evolved individual is imposing it to the receiving subpopulation. This problem is known as the Conquest Problem [27].

Symmetrically, when a less evolved individual migrates to a highly evolved subpopulation it will not be selected for reproduction and therefore it will be abandoned. This means a waste of computational and communication efforts. This problem is known as the Non-effect Problem [27].

Both problems could appear in DGAs since subpopulations tend to converge at different speeds. For example, if parameters used for the genetic operators are different, convergence speed will be very different in subpopulations. These problems can directly affect the global convergence leading to non-optimal solutions and losing the effectiveness of the distributed approach.

Subsequently, proposing a well-designed DGA is not a trivial task due to the existence of several factors that can have an influence over the exploration/exploitation balance of the algorithm. There are several elements to consider when designing DGAs:

1. Topology: structure of the distributed algorithm which defines relationships between subpopulations and individuals [38, 41–43].
2. Migration rate (MRATE): amount of individuals to be exchanged between subpopulations.
3. Migration frequency (MFREQ): number of generations between two consecutive migrations.

4. Selection strategy: generally there are two ways of selecting the genetic material to be copied. The first way is randomly selecting an individual from the current subpopulation. The second way consists on selecting the individual with the best fitness in every subpopulation to be copied to another. The last one would lead into a more direct evolution because individuals would not have traces of less adapted individuals. The main disadvantage of selecting the best individual is that it could lead into premature convergence [44].
5. Replacement strategy: different replacement strategies can be considered, as replacing the worst individuals with the ones received due to migrations, as replacing an individual randomly choosen, etc.
6. Replication of emigrants: should individuals be moved, or copied among subpopulations? Exchanging copies of individuals could lead to a highly evolved individuals dominating several less evolved subpopulations [44].

All these parameters have a deep interaction among them and should be carefully determined since a poor choice in one of them can have a strong impact on the global performance of the algorithm. For instance, choosing an extremely high MFREQ can lead to an excessive communication load of the network and the effect of the migrated individuals could be almost imperceptible. Besides, these parameters should be fixed having in mind the hardware that will be used to execute the algorithm: depending on the network it might be better migrating more individuals less frequently than the other way around.

Finally, the DGA schema that comes from the consideration of spatial separation of subpopulations is presented in Table 1.

Table 1 DGA schema

1. Generate a random population, P
2. Divide P into m subpopulations: SPi, i = 1, …, m
3. Define a topology for SP1, …, SPm
4. For i = 1 to m do
4.1. Apply in parallel during MFREQ generations the genetic operators
4.2. Send in parallel MRATE chromosomes to neighbour subpopulations
4.3. Receive in parallel chromosomes from neighbour subpopulations
5. If stopping criteria is not meet then go back to step 4

3 Genetic tuning of FRBSs: a particular case on the effective lateral tuning of MFs

With the aim of making an FRBS perform better, some approaches try to improve the preliminary DB definition or the inference engine parameters once the RB has been derived [9–11]. In order to do so, a tuning process considering the whole KB obtained (the preliminary DB and the derived RB) is used a posteriori to adjust the MFs or the inference engine parameters. A graphical representation of the tuning process is shown in Fig. 2.

Among the different possibilities to perform the tuning, one of the most widely-used approaches to enhance the performance of FRBSs is the one focused on the DB definition, usually named *tuning of MFs*, or *DB tuning* [15, 17–19, 45]. In [19], we can find a first and classic proposal on the tuning of MFs. In this case, the tuning methods refine the parameters that identify the MFs associated to the labels comprising the DB. Classically, due to the wide use of the triangular-shaped MFs, the tuning methods [10, 17–19] refine the three definition parameters that identify these kinds of MFs (Fig. 3).

Since the parameters of the MF are interdependent among themselves, in the case of large scale problems, the tuning process becomes an optimization problem on a very complex search space. This, of course, affects the good performance of the optimization methods. A good alternative to solve this problem is the lateral tuning of MFs [15]. This approach makes use of the linguistic 2-tuples representation [28] which simplifies the search space and, therefore, eases the derivation of optimal models, particularly in complex or high-dimensional problems.

In order to better handle the complex search space that the tuning of MFs represents, in this work, we analyze the use of the DGAs combined with a local parallelization when performing a lateral tuning of the MFs.

In Sect. 3.1, we describe the efficient lateral tuning of FRBSs. Next, the sequential evolutionary algorithm proposed in [15] to perform the lateral tuning of FRBS is briefly described in Sect. 3.2.

3.1 Rules with the linguistic 2-tuples representation

In [15], a new procedure for FRBSs tuning was proposed. It is based on the linguistic 2-tuples representation scheme introduced in [28], which allows the lateral displacement of the support of a label and maintains the interpretability at a good level. This proposal introduces a new model for rule representation based on the concept of symbolic translation [28]. The symbolic translation of a label is a number in $(-0.5, 0.5)$ which expresses its displacement between two adjacent lateral labels (Fig. 4a). Let us consider a generic linguistic fuzzy partition $S = \{s_0,\ldots, s_{L-1}\}$ (with L

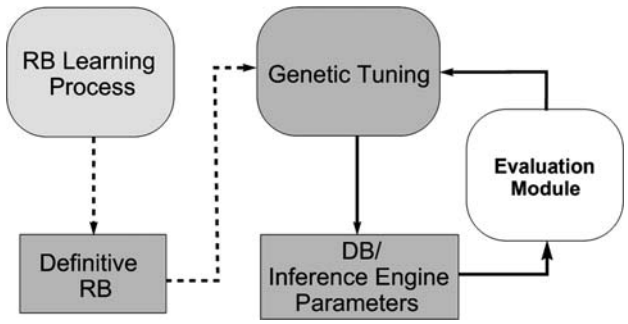

Fig. 2 Genetic tuning process

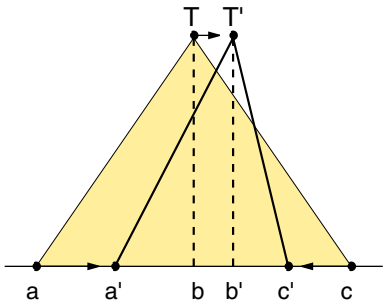

Fig. 3 Tuning by changing the basic MF parameters

representing the number of labels). Formally, we represent the symbolic translation of a label s_i in S by means of the 2-tuple notation,

$$(s_i, \alpha_i), \quad s_i \in S, \quad \alpha_i \in [-0.5, 0.5).$$

The symbolic translation of a label involves the lateral variation of its associated MF. Figure 4 shows the symbolic translation of a label represented by the 2-tuple (s_2, -0.3) together with the associated lateral variation.

In the context of FRBSs, the linguistic 2-tuples could be used to represent the MFs used in the linguistic rules. This way to work, introduces a new model for rule representation that allows the tuning of the MFs by learning their respective lateral displacements. Next, we present this approach by considering a simple control problem.

Let us consider a control problem with two input variables (X_1, X_2), one output variable (Y) and an initial DB defined by experts to determine the MFs for the following labels:

- X_1: *Error* $\rightarrow$ {*Negative*, *Zero*, *Positive*}
- X_2: ∇*Error* $\rightarrow$ {*Negative*, *Zero*, *Positive*}
- Y: *Power* $\rightarrow$ {*Low*, *Medium*, *High*}

Based on this DB definition, examples of classic and linguistic 2-tuples represented rules are:

- Classic Rule:
  R_i: If the *Error* is Zero and the ∇Error is Positive Then the *Power* is High.
- Rule with 2-Tuples Representation:
  R_i: If the *Error* is (Zero,0.3) and the ∇Error is (Positive, -0.2) Then the *Power* is (High, -0.1).

With respect to the classic tuning, usually considering three parameters in the case of triangular MFs, this way to work involves a reduction of the search space that eases a fast derivation of optimal models, improving the convergence speed and avoiding the necessity of a large number of evaluations.

In [15], two different rule representation approaches have been proposed, a global approach and a local approach. The global approach tries to obtain more interpretable models, while the local approach tries to obtain more accurate ones. In our case, tuning is applied at the level of linguistic partitions (global approach). By considering this approach, the label s_i^v of a variable v is translated with the same α_i^v value in all the rules where it is used, i.e., a global collection of 2-tuples is used in all the fuzzy rules.

Notice that from the parameters α_i^v applied to each label we could obtain the equivalent triangular MFs. Thus, an FRBS based on linguistic 2-tuples can be represented as a classic Mamdani FRBS [46]. Refer to [15] for further details on this approach.

3.2 Lateral tuning of FRBSs

In [15], two effective sequential GAs were respectively proposed to perform a lateral tuning, or to combine it with a rule selection, on previously obtained FRBSs. In this paper, we will focus on the first one, only performing tuning, as a way to analyze the aplicability of DGAs for the tuning of FRBSs. A short description of this algorithm is given below (see ref. [15] for a detailed description).

As the basis optimization procedure the genetic model of CHC [47] was used. The evolutionary model of CHC makes use of a "Population-based Selection" approach. N parents and their corresponding offsprings are combined to select the best N individuals to compose the next population. The CHC approach makes use of an incest prevention mechanism and a restarting process to provoke diversity in the population, instead of the well known mutation operator.

This incest prevention mechanism is considered in order to apply the crossover operator, i.e., two parents are crossed if their hamming distance divided by 2 is higher than a predetermined threshold, T. Since a real coding scheme is considered, each gene is transformed by considering a Gray Code with a fixed number of bits per gene (*BITSGENE*) determined by the system expert. In our case, the threshold value is initialized as:

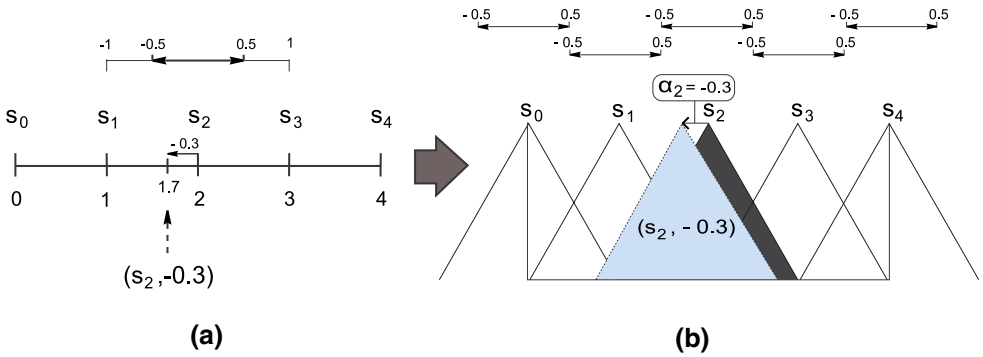

Fig. 4 Symbolic translation of a label and lateral displacement of the associated MF. *a* Symbolic translation of a label; *b* lateral displacement of a membership function

$$T = (\#GenesC_T * BITSGENE)/4.0.$$

Following the original CHC scheme, T is decreased by one when the population does not change in one generation. In order to avoid very slow convergence, T is also decreased by one when no improvement is achieved with respect to the best chromosome of the previous generation. The algorithm restarts when T is below zero. A scheme of the evolutionary model of CHC is shown in Fig. 5.

In the following, the components used to design the evolutionary tuning process are explained. They are: DB codification and initial gene pool, fitness function, crossover operator and restarting process.

3.2.1 Data base codification and initial population

A real coding scheme is considered, i.e., the real parameters are the GA representation units (genes). Let us consider n system variables and a fixed number of labels per variable L. Then, a chromosome has the following form (where each gene is associated to the tuning value of the corresponding label),

$$(\alpha_1^1, \ldots, \alpha_1^L, \alpha_2^1, \ldots, \alpha_2^L, \ldots, \alpha_n^1, \ldots, \alpha_n^L).$$

To make use of the available information, the initial FRBS obtained from an automatic fuzzy rule learning method is included in the population as an initial solution. To do so, the initial pool is obtained with the first individual having all genes with value '0.0', and the remaining individuals generated at random in $(-0.5, 0.5)$.

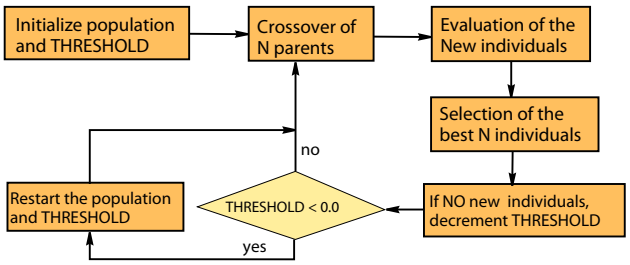

Fig. 5 Scheme of CHC

3.2.2 Fitness function

To evaluate a given chromosome the well-known Mean Square Error (MSE) is used:

$$\text{MSE} = \frac{1}{2 \cdot N} \sum_{l=1}^{N} (F(x^l) - y^l)^2,$$

with N being the data set size, $F(x^l)$ being the output obtained from the FRBS decoded from the said chromosome when the l-th example is considered and y^l being the known desired output.

3.2.3 Crossover operator

The crossover operator is based on the the concept of environments. These kinds of operators show a good behavior in real coding. Particularly, the BLX-α operator [48] is considered.

This operator allows tuning the degree of exploration and exploitation of the crossover in an easy way. BLX-α crossover works as follows: let us assume that $X = (x_1,\ldots, x_g)$ and $Y = (y_1,\ldots, y_g)$ with $x_i, y_i \in (a_i, b_i) = [-0.5, 0.5) \subset \mathbb{R} (i = 1, \ldots, g)$ are the two real-coded chromosomes that are going to be crossed. Using the BLX-α crossover, one descendant $Z = (z_1,\ldots, z_g)$ is obtained, where z_i is randomly (uniformly) generated within the interval $[l_i, u_i]$, with $l_i = max\{a_i, c_{min} - A\}$, $u_i = min\{b_i, c_{max} + A\}$, $c_{min} = min\{x_i, y_i\}$, $c_{max} = max\{x_i, y_i\}$ and $A = (c_{max} - c_{min})\cdot\alpha$.

Figure 6 shows how the BLX-α operator works at different stages of the evolution process (convergence to a common point) with $\alpha = 0.5$ as an example of the behavior of this operator. Even though that a change on the alpha value promotes different speeds to change from exploration to exploitation, the shown stages are present in the evolution for any value of alpha.

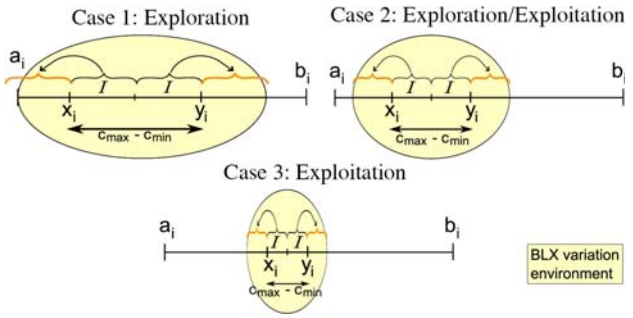

Fig. 6 Different cases/stages of the application of the BLX-α crossover operator, where $\alpha = 0.5$

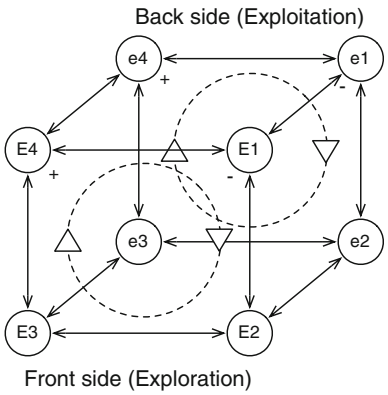

Fig. 7 Hypercube topology for GDRCGA

3.2.4 *Restarting process*

To get away from local optima, this algorithm uses a restart approach [47]. In this case, the best chromosome is maintained and the remaining are generated at random within the corresponding variation intervals $(-0.5, 0.5)$. It follows the principles of CHC [47], performing the restart procedure when the threshold T is below zero.

4 A distributed genetic algorithm for the lateral tuning of FRBSs

One of the problems when performing tuning with complex data sets is the complexity of the search space. Sometimes even an advanced GA can not deal with the complex search space in terms of time and quality of the results.

GDRCGAs are a kind of heterogeneous DGAs based on real coding where subpopulations apply genetic operators in different levels of exploitation/exploration. This heterogeneous application of genetic operators produce a *parallel multiresolution* which allows a wide exploration of the search space and effective local precision. Due to appropiate connections between subpopulations in order to gradually exploit multiresolution, these algorithms achieve refinement or expansion of the best emerging zones of the search space.

In order to analyze how DGAs can help the tuning problem, we have selected an efficient GDRCGA [27], that keeps a good balance between exploration and exploitation of the search space. additionally, we have combined this GDRCGA (for global parallelization) with a local parallelization to perform fitness evaluations in a parallel way at each subpopulation. As we said before, we apply this algorithm, namely GDRCGA, to perform a lateral tuning of previously obtained FRBSs.

This section is organized as follows. Section 4.1 sets out the main components of the DGA such as topology, migrations scheme, etc. Next, Sect. 4.2 explains the common components of the different subpopulations such as crossover operator, DB codification, etc.

4.1 Main components of the DGA

The GDRCGA [27] used for FRBS tuning employs 8 subpopulations in a hypercube topology as seen in Fig. 7.

In this topology two important groups of subpopulations can be clearly identified:

1. Front side: this side of the hypercube is oriented to explore the search space. In this side, four subpopulations, $E_1,\ldots, E_4$, apply genetic operators adapted for exploration in a clockwise increasing degree.
2. Back side: subpopulations in the back side of the hypercube, $e_1,\ldots, e_4$, apply exploitation oriented genetic operators in a clockwise increasing degree.

One of the key elements of DGAs is the migration policy of individuals between subpopulations. In this particular model, an *immigration* process [49] is achieved since the best chromosome in every subpopulation abandons it and moves to an immediate neighbor. Due to this immigration policy, three different immigration movements can be identified depending on the subpopulations involved:

1. *Refinement migrations*: individuals in the back side move clockwise to the immediate neighbor, i.e. from e_2 to e_3. Chromosomes in the front side move counterclock from a more exploratory subpopulation to a less exploratory oriented one.
2. *Expansion migrations*: individuals in the back side move counterclock to the immediate neighbor and chromosomes in the front side move clockwise from a less exploratory subpopulation to a more exploratory oriented one, i.e. from E_4 to E_1.
3. *Mixed migrations*: subpopulations from one side of the hypercube exchange their best individual with the

counterpart subpopulation in the other side: interchange between E_i and e_i, $i = 1\ldots4$.

Figure 8 shows the three different migration movements described above.

As stated in [27], the frequency in which migration movements occur is crucial to avoid the classic withdraws of DGAs: the conquest and noneffect problems. In order to reduce the negative effect of these problems, immigrants stay in the receiving subpopulations for a brief number of generations. Besides, a global restarting operator is used to avoid stagnation of the search process. This restart operator randomly reinitializes all subpopulations if non-significant improvement of the best element is achieved for a number of generations. Also an elitism strategy is used in order to keep the best adapted individual of every subpopulation.

4.2 Common components of individual subpopulations

The main component used in the different subpopulations of the distributed model are:

- DB codification and initial subpopulations: the coding scheme used to represent the displacement parameters is the same one described in Sect. 3.2.1 for the specialized sequential algorithm. Each subpopulation is also initialized in the same way explained in Sect. 3.2.1, i.e., by including the initial FRBS as the first individual in each subpopulation and the remaining individuals generated at random.
- Crossover operator: the crossover operator used, BLX-α, is the same that was used in the specialized sequential algorithm and is described in Sect. 3.2.3. As stated before, distinct parameter values are used between subpopulations in order to achieve different degrees of exploitation/exploration. The values used for each subpopulation are shown in Table 2.

In the absence of selection pressure, values of α, which are $\alpha < 0.5$ make the subpopulations converge towards values in the center of their ranges, producing low diversity levels in the population and inducing a possible premature convergence towards non-optimal solutions. Only when $\alpha = 0.5$, there is a balanced relationship reached between convergence (exploitation) and divergence (exploration). In this case, the probability that a gene will lie in the exploration interval is equal to the probability that it will lie in an exploration interval [48].

- Local restarting process: In each subpopulation the same restarting approach explained in Sect. 3.2.4 is used, i.e., the best chromosome is maintained and the remaining ones are generated at random within the

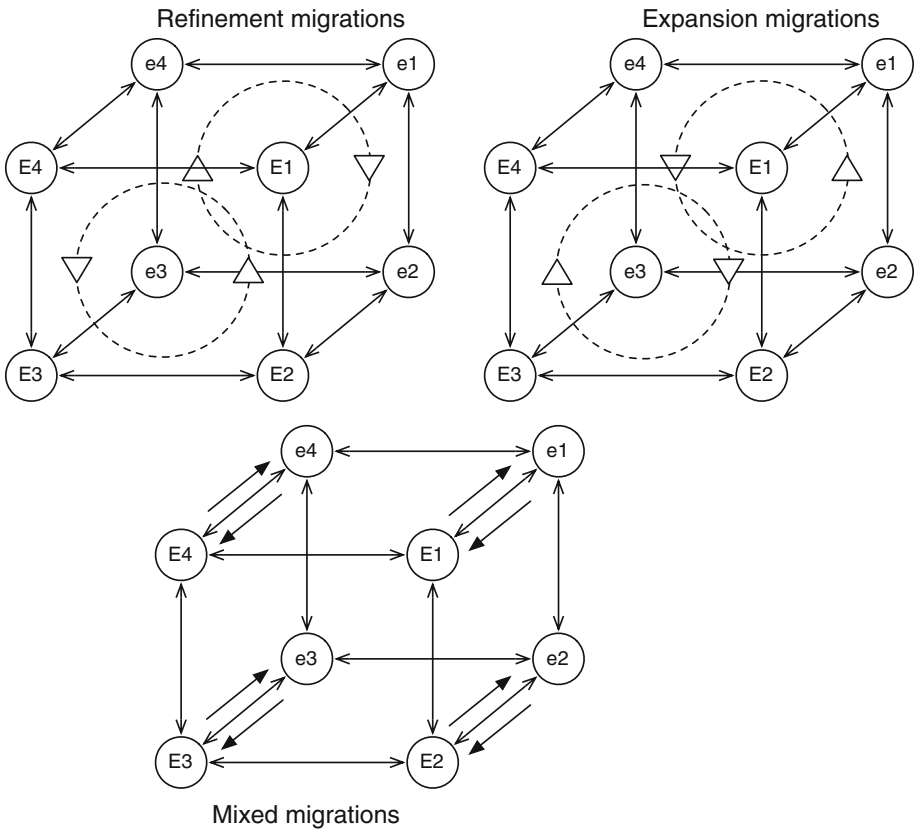

Fig. 8 Three different migration movements for the GDRCGA

Table 2 Values of α for each subpopulation

Exploitation $+ \leftarrow -$				Exploration $- \rightarrow +$			
e_4	e_3	e_2	e_1	E_1	E_2	E_3	E_4
0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8

corresponding variation intervals $(-0.5, 0.5)$ when each particular threshold T is below zero. Therefore, each subpopulation has its own local restating operator.

- Local parallelization for fitness evaluation: with the aim of improving execution times for the distributed algorithm, we use several threads to perform the fitness evaluation of individuals within each subpopulation. This multi-thread approach is quite straightforward: every subpopulation is divided into a number of groups and every group is evaluated using a different thread. Generally, an excellent number of groups is the number of cores present in every processor of the hardware architecture.

5 Experiments

To evaluate the distributed approach, we have addressed nine real-world problems with different complexities (different numbers of variables and available data). Table 3 summarizes the main characteristics of the nine datasets and shows the link to the KEEL project webpage [50] from which they can be downloaded. This section is organized as follows:

- First, we describe the experimental set-up in Sect. 5.1.
- Second, we compare the different approaches, sequential and distributed in terms of performance, in Sect. 5.2.
- Third, we study the convergence and trend of the compared algorithms in Sect. 5.3.
- Finally, a speed-up analysis is performed in order to show the computational gains obtained by the distributed approach in Sect. 5.4.

Table 3 Data sets considered for the experimental study

Datasets	Name	Variables	Examples
Electrical length	EL	2	495
Electrical maintenance	EM	4	1,056
Abalone	ABA	8	4,177
Stock	ST	9	950
Weather-Izmir	WI	9	1,461
Weather-Ankara	WA	9	1,609
Treasury	TR	15	1,049
Mortgage	MOR	15	1,049
Computer activity	CA	21	8,192

Available at http://sci2s.ugr.es/keel/datasets.php

5.1 Experimental set-up

We applied the algorithms described in the previous sections both the sequential and distributed approaches, namely, *CHC* and *GDRCGA*. Table 4 summarizes the configurations of the algorithms used in the experiments.

In all the cases, the well-known *ad-hoc* data-driven learning algorithm of Wang and Mendel [51] (WM-Method) is applied to obtain an initial set of candidate linguistic rules. The initial linguistic partitions are comprised of *five linguistic terms* in the case of datasets with less than 9 variables and *three linguistic terms* in the remaining ones. We consider strong fuzzy partitions of triangular-shaped MFs. Once the initial RB is generated, the different algorithms can be applied. The performance of the initial FRBSs obtained by WM-Method are shown in Table 5 (initial results taken as a reference).

We adopted a 5-fold cross-validation model, i.e., we randomly split the data set into 5 folds, each containing the 20% of the examples of the data set, and used four folds for training and one for testing. For each dataset we compute the mean values of MSE in training and test sets.

In order To assess whether significant differences exist among results, we adopt statistical analysis and in particular non-parametric tests [29–32], according to the recommendations made in [29, 31], where a set of simple, safe and robust non-parametric tests for statistical comparisons of classifiers have been introduced. In particular, we use the Wilcoxon signed-rank test [52, 53] for pairwise comparison of the two algorithms. A detailed description of this test is included in Appendix. To perform the test we use a level of confidence of $\alpha = 0.1$. This statistical test is based on computing the differences on two sample means (typically, mean test errors obtained by a pair of different algorithms on different data sets). In the classification framework these differences are well defined since these errors are in the same domain. In the regression framework, to have well defined differences, we propose to adopt a normalised difference *DIFF*, defined as:

$$\text{DIFF} = \frac{\text{MSE}_{\text{tst}}(\text{CHC}) - \text{MSE}_{\text{tst}}(\text{GDRCGA})}{\text{MSE}_{\text{tst}}(\text{CHC})}. \quad (1)$$

This difference expresses the improvement percentage of the distributed algorithm over the sequential one.

Table 4 Algorithm configurations used in the experiments

Parameter	CHC	GDRCGA
Evolution approach	CHC	CHC
Elitism	Yes	Yes
Total evaluations	100,000	100,000
Crossover operator	BLX −0.5	Gradual BLX −α
Restart	Yes	Yes
Topology	Single population	Hypercube
Migration frequency	–	5 Generations
Size of population (CHC)/sub-populations (GDRCGA)	50	50

Table 5 Initial results obtained by WM-method

Dataset	MSE_{tra}	MSE_{tst}
EL	2.347E+05	2.419E+05
EM	5.761E+04	5.793E+04
ABA	8.407E+00	8.422E+00
ST	9.074E+00	9.042E+00
WI	6.944E+00	7.368E+00
WA	1.606E+01	1.639E+01
TR	1.636E+00	1.631E+00
MOR	9.850E−01	9.730E−01
CA	4.038E+01	4.096E+01

5.2 Performance analysis

Table 6 shows the average results corresponding to the solutions obtained for the nine datasets and the two algorithms compared. In the table, MSE_{tra}, MSE_{tst} and DIFF denote, respectively, the MSE on the training set, the MSE on the test set and the improvement value defined in Eq. 1.

We observe that the distributed algorithm GDRCGA outperforms the sequential algorithm in eight out of the nine datasets in both training and test. To assess whether we can conclude that GDRCGA statistically outperforms CHC in terms of MSEs, we apply the Wilcoxon signed-rank test to the results achieved by these algorithms. Table 7 shows the results of the application of the Wilcoxon test on the test set. Here, R^+ and R^- denote, respectively, the sum of the ranks corresponding to CHC and GDRCGA. The null hypothesis associated with the Wilcoxon signed-rank test is rejected in favour of GDRCGA due to the differences between R^+ and R^-. Thus, we can conclude that the results achieved by these algorithms are statistically different on the test set.

5.3 Convergence study and trend of both algorithms

The observation of the evolution of the MSE is also an interesting factor to take into account. Two different data sets have been choosen in order to study the evolution of the MSE: Electrical Maintenance and Treasury. These two data sets were choosen because of their different complexity: Treasury data set is far more complex than Electrical Maintenance.

Figure 9 shows the convergence of both algorithms in both problems. Due to the distributed nature of the algorithm and consequently the spatial separation implied, in the more complex problem, it needs more evaluations to converge than the sequential algorithm. It always presents the same behaviour in comparision to the sequential approach: with a small number of evaluations it yields a higher error than the sequential one in the most complex problems, but when the number of evaluations is high it gives solutions with a better quality.

As it has been stated, the distributed approach needs more iterations to achieve convergence for complex data sets. This situation can be observed in Figure 9, right side: GDRCGA achieves better MSE values when the search process has consumed almost two thirds of the number of evaluations. On the other hand, when dealing with less complex data sets like Electrical Maintenance (Fig. 9, left side), the distributed approach quickly achieves better MSE values from almost the begining of the search process and keeps gaining distance from the sequential CHC algorithm. In fact, GDRCGA begins achieving better MSE values shortly after the search process has consumed less than half of the number of evaluations available. These two situations can be also verified in Table 6.

Besides studying the evolution of the MSE in training (convergence), it is also interesting to analyze the effects that it produces on the MSE in test regarding the same data sets. Figure 10 shows the MSE in test of Treasury and Electrical Maintenance datasets. Again, we can observe that the distributed approach needs more evaluations to outperform the sequential algorithm in the more complex problem (Fig. 10, right side) while better results are obtained practically from the beginning in the simpler one (Fig. 10, left side). However, two interesting characteristics can be highlighted. Firstly, the evolution in the test error shown by GDRCGA seems more stable in both problems. Secondly, GDRCGA shows practically the same trend in training and test in both datasets, while the sequential approach worsen the test error once the half of the evaluations are consumed in the more complex dataset (overfitting). Actually, the delay in convergence helps it to escape from overfitting and locate better optima. Both characteristics are quite recommendable in the fuzzy modeling framework.

Table 6 Experimental results obtained by CHC and GDRCGA

Data set	Evaluations	CHC		GDRCGA		DIFF
		MSE_{tra}	MSE_{tst}	MSE_{tra}	MSE_{tst}	
EL	25,000	1.672E+05	1.932E+05	1.665E+05	1.910E+05	1.156E−02
	50,000	1.672E+05	1.930E+05	1.593E+05	1.867E+05	3.268E−02
	100,000	1.672E+05	1.928E+05	1.332E+05	1.824E+05	5.375E−02
EM	25,000	2.482E+04	2.807E+04	2.381E+04	2.729E+04	2.777E−02
	50,000	2.464E+04	2.796E+04	2.270E+04	2.703E+04	3.328E−02
	100,000	2.394E+04	2.749E+04	2.258E+04	2.682E+04	2.447E−02
ABA	25,000	2.616E+00	2.802E+00	2.600E+00	2.782E+00	7.050E−03
	50,000	2.616E+00	2.802E+00	2.568E+00	2.747E+00	1.961E−02
	100,000	2.616E+00	2.802E+00	2.444E+00	2.736E+00	2.349E−02
ST	25,000	4.600E−01	6.332E−01	4.592E−01	6.352E−01	−3.209E−03
	50,000	4.549E−01	6.307E−01	4.347E−01	6.292E−01	2.424E−03
	100,000	4.548E−01	6.291E−01	4.289E−01	6.276E−01	2.402E−03
WI	25,000	1.643E+00	1.868E+00	1.679E+00	1.883E+00	−8.030E−03
	50,000	1.635E+00	1.863E+00	1.599E+00	1.854E+00	4.803E−03
	100,000	1.444E+00	1.851E+00	1.393E+00	1.702E+00	8.059E−02
WA	25,000	2.720E+00	3.531E+00	2.621E+00	3.627E+00	−2.712E−02
	50,000	2.683E+00	3.429E+00	2.578E+00	3.012E+00	1.217E−01
	100,000	2.621E+00	3.367E+00	2.445E+00	3.166E+00	5.973E−02
TR	25,000	1.342E−01	1.514E−01	1.431E−01	1.662E−01	−9.750E−02
	50,000	1.218E−01	1.374E−01	1.162E−01	1.321E−01	3.832E−02
	100,000	1.147E−01	1.304E−01	1.106E−01	1.224E−01	6.183E−02
MOR	25,000	1.070E−01	1.207E−01	1.292E−01	1.341E−01	−1.113E−01
	50,000	1.057E−01	1.198E−01	1.218E−01	1.324E−01	−1.052E−01
	100,000	1.029E−01	1.177E−01	1.152E−01	1.287E−01	−9.358E−02
CA	25,000	4.528E+00	5.534E+00	4.562E+00	5.712E+00	−3.214E−02
	50,000	4.515E+00	5.673E+00	4.518E+00	5.692E+00	−3.426E−03
	100,000	4.505E+00	5.662E+00	4.419E+00	5.595E+00	1.179E−02

Table 7 Wilcoxon test to compare CHC with GDRCGA

Comparison	R^+	R^-	Hypothesis ($\alpha = 0.1$)	p-value
CHC versus GDRCGA (100000 evs.)	8.0	37.0	Rejected	0.086

R^+ corresponds to CHC and R^- to GDRCGA

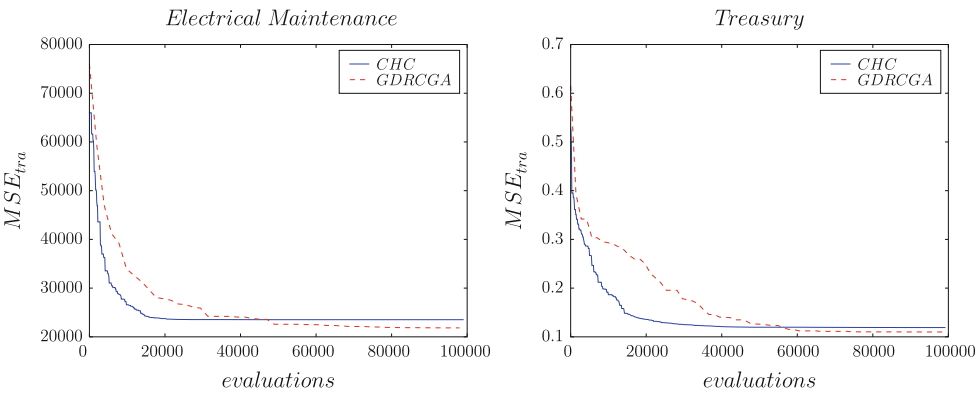

Fig. 9 Evolution of the MSE in training (convergence): electrical maintenance and treasury datasets

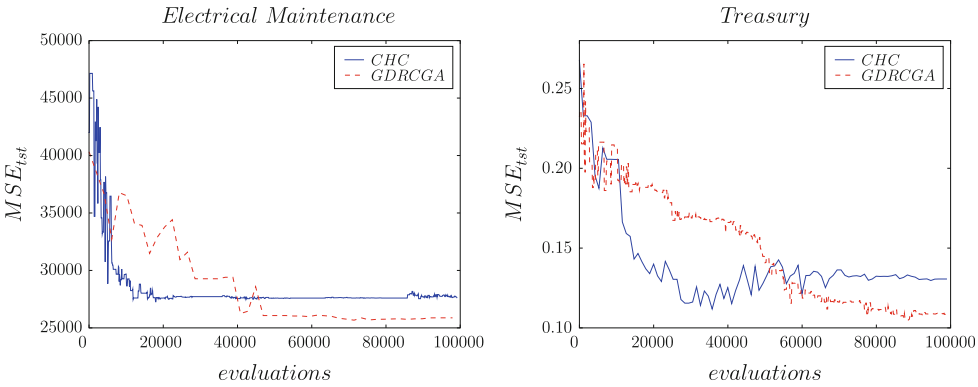

Fig. 10 Effects on the MSE in test: electrical maintenance and Treasury data sets

5.4 Speed-up study

Generally, when comparing a distributed or parallel approach with the corresponding sequential algorithm an interesting measure is the execution time gain ratio, also known as *speed-up*. This ratio could be defined as follows:

$$S_{\text{up}} = \frac{T_{\text{seq}}}{T_{\text{dist}}}, \tag{2}$$

where T_{seq} is the time spent by the sequential algorithm and T_{dist} is the execution time of the distributed approach. This measure expresses how much faster the distributed approach is compared to the sequential one. The higher the value of S_{up}, the better our approach performs. To compute the mean *speed-up* of the distributed algorithm, we have considered the average execution times for each dataset and algorithm. We used a cluster with 8 nodes, each one with 8 GB RAM and an Intel Core 2 Quad Q9300 processor at 2.5 GHz. As said, individuals within each subpopulation were evaluated in parallel, four individuals at a time, taking advantage of the four *cores* present in every processor of the cluster.

Table 8 shows mean *speed-up* values obtained for the nine data sets along with the number of variables, the number of examples and the number of variables multiplied by the number of examples (VE). The data sets are sorted in increasing speed-up gains order. In the less complex data sets the speed-up obtained is substantially lower because the sequential algorithm is very fast and the time spent in communications of the distributed approach slows it down in comparison. As the complexity of the data set increases the speed-up also increases, showing that the distributed approach in the most complex data set is more than five and a half times faster than the sequential algorithm. The distributed algorithm takes longer than the sequential algorithm when dealing with small size data sets mainly due to two reasons: interprocess communication in the distributed approach implies additional execution time which can not be parallelized and the specialized algorithm is optimized for small size data sets where the search space is not too complex. The distributed algorithm takes adventage of the slow individual evaluation performed by the sequential algorithm in complex search spaces which makes it suitable for dealing with problems where fitness evaluation is computationally difficult.

Table 8 Speed-up values obtained in increasing speed-up gains order

Datasets	Variables	Examples	VE	Speed-up
EL	2	495	990	0.49
EM	4	1,056	4,224	0.51
ST	9	950	8,550	0.63
WI	9	1,461	13,149	1.07
WA	9	1,609	14,481	1.12
TR	15	1,049	15,735	1.81
MOR	15	1,049	15,735	1.92
ABA	8	4,177	33,416	2.13
CA	21	8,192	172,032	5.61

Figure 11 represents the trend of the values shown in Table 7 for the number of variables, examples and VE. It can be seen how the speed-up is not independently related to the number of variables or examples but it is related to both together.

6 Concluding remarks

In this paper we have presented a study on the use of the DGAs for the lateral tuning of FRBSs. To this end, we have analyzed the performance of a specific GDRCGA employing 8 subpopulations in a hypercube topology [26] together with a local parallelization of the fitness evaluation at each subpopulation. This algorithm has been compared with the specialized GA presented in [15] to perform the lateral tuning of FRBSs.

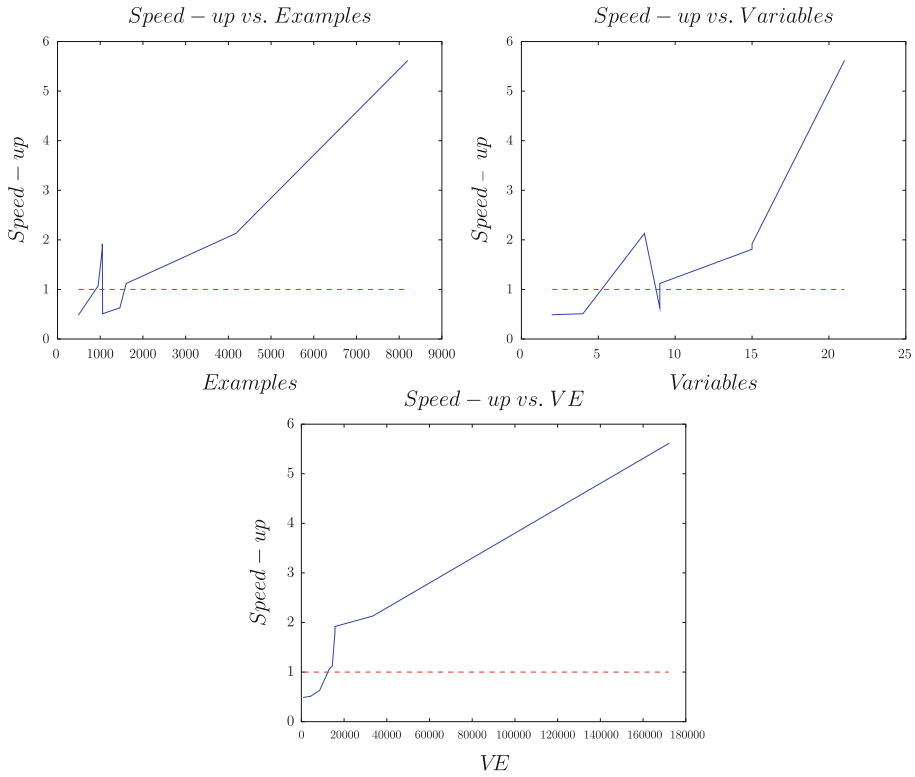

Fig. 11 Influence of the number of examples, variables and VE on the speed-up

From the empirical results obtained, we can conclude that as the complexity of the problem grows, the distributed approach outperforms the specialized sequential algorithm. Moreover, the distributed procedure makes effective use of the wall time in relation to the computing times required by the sequential algorithm. Also, when dealing with complex search spaces, the distributed approach is able to converge to better quality solutions than the sequential algorithm. This behaviour makes the distributed tuning algorithm very useful when dealing with large scale problems where the complexity of the search space is high.

Since execution time and quality of the results are two properties always in conflict somehow, the distributed approach could be graduated in order to achieve faster execution times with a small cost in quality and vice versa.

Acknowledgments This work was supported by the Spanish Ministry of Science and Innovation under grant TIN2005-08386-C05-01. Authors would like the thank the UGRGrid team from the University of Granada for their continuous support.

Appendix: Wilcoxon's Signed-Rank Test

The Wilcoxon signed-rank test is a pair-wise test that aims to detect significant differences between two sample means: it is the analogous to the paired t-test in non-parametric statistical procedures. If these means refer to the outputs of two algorithms, then the test practically assesses the reciprocal behavior of the two algorithms [52, 53]. Let d_i be the difference between the performance scores of the two algorithms on the i-th out of N_{ds} datasets. The differences are ranked according to their absolute values; average ranks are assigned in case of ties. Let R^+ be the sum of ranks for the datasets on which the first algorithm outperformed the second, and R^- the sum of ranks for the contrary outcome. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored:

$$R^+ = \sum_{d_i > 0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i),$$

$$R^- = \sum_{d_i < 0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i).$$

Let T be the smaller of the sums, $T = min(R^+, R^-)$. If T is less than, or equal to, the value of the distribution of Wilcoxon for N_{ds} degrees of freedom (Table B.12 in [54]), the null hypothesis of equality of means is rejected.

The Wilcoxon signed-rank test is more sensible than the t-test. It assumes commensurability of differences, but only qualitatively: greater differences still count for more, which is probably desired, but the absolute magnitudes are ignored. From the statistical point of view, the test is safer since it does not assume normal distributions. Also, the outliers (exceptionally good/bad performances on a few datasets) have less effect on the Wilcoxon test than on the t-test. The Wilcoxon test assumes continuous differences d_i, therefore they should not be rounded to one or two

decimals, since this would decrease the test power due to a high number of ties.

When the assumptions of the paired t-test are met, the Wilcoxon signed-rank test is less powerful than the paired t-test. On the other hand, when the assumptions are violated, the Wilcoxon test can be even more powerful than the t-test. This allows us to apply it to the means obtained by the algorithms in each dataset, without any assumption about the distribution of the obtained results.

References

1. Driankow D, Hellendoorn H, Reinfrank M (1993) An introduction to fuzzy control. Springer, Berlin
2. Ishibuchi H, Nakashima T, Nii M (2004) Classification and modeling with linguistic information granules: advances approaches to linguistic data mining. Springer, Berlin
3. Palm R, Driankov D, Hellendoorn (1997) Model based fuzzy control. Springer, Berlin
4. Pedrycz W (1996) Fuzzy modelling: paradigms and practice. Kluwer, Norwell
5. Zadeh LA (1965) Fuzzy sets. Inf Control 8: 338–353
6. Zadeh LA (1973) Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans Syst Man Cybern 3: 28–44
7. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, New York
8. Holland JH (1992) Adaptation in natural and artificial systems (The University of Michigan Press 1975). MIT, London
9. Cordón O, Gomide F, Herrera F, Hoffmann F, Magdalena L (2004) Ten years of genetic fuzzy systems: current work and new trends. Fuzzy Sets Syst 141(1): 5–31
10. Cordón O, Herrera F, Hoffmann F, Magdalena L (2001) Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases. World Scientific, Singapore
11. Herrera F (2008) Genetic fuzzy systems: taxonomy, current research trends and prospects. Evol Intell 1: 27–46
12. Eiben AE, Smith JE (2003) Introduction to evolutionary computation. Springer, Berlin
13. Zadeh LA (1975) The concept of a linguistic variable and its applications to approximate reasoning, parts i, ii and iii. Inf Sci 8(8 and 9):199–249, 301–357, 43–80
14. Alcalá R, Alcalá-Fdez J, Casillas J, Cordón O, Herrera F (2006) Hybrid learning models to get the interpretability-accuracy trade-off in fuzzy modeling. Soft Comput 10(9):717–734
15. Alcalá R, Alcalá-Fdez J, Herrera F (2007) A proposal for the genetic lateral tuning of linguistic fuzzy systems and its interaction with rule selection. IEEE Trans Fuzzy Syst 15(4):616–635
16. Casillas J, Cordón O, del Jesus MJ, Herrera F (2003) Accuracy improvements in linguistic fuzzy modeling. Springer, Berlin
17. Casillas J, Cordón O, del Jesus MJ, 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
18. Herrera F, Lozano M, Verdegay JL (1995) Tuning fuzzy logic controllers by genetic algorithms. Int J Approx Reason 12:299–315
19. Karr C (1991) Genetic algorithms for fuzzy controllers. AI Expert 6(2):26–33
20. Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley, New York
21. Cantu-Paz E (2000) Efficient and accurate parallel genetic algorithms. Kluwer, Norwell
22. de Vega FF, Cantu-Paz E (2008) Special issue on distributed bioinspired algorithms. Soft Comput 12(12):1143–1144
23. Dowd K, Severance C (1998) High performance computing. O'Reilly, Sebastopol
24. Spector DHM (2000) Building Linux clusters. O'Reilly, Sebastopol
25. Sterling T, Becker DJ, Savarese DF (1999) How to build a beowulf: a guide to the implementation and application of PC clusters. MIT, Cambridge
26. Robles I, Alcalá R, Benítez JM, Herrera F (2009) Distributed genetic tuning of fuzzy rule-based systems. In: Proceedings of the international fuzzy systems association—European society for fuzzy logic and technology (IFSA-EUSFLAT) congress (in press)
27. Herrera F, Lozano M (2000) Gradual distributed real-coded genetic algorithms. IEEE Trans Evol Comput 4(1): 43–63
28. Herrera F, Martínez L (2000) A 2-tuple fuzzy linguistic representation model for computing with words. IEEE Trans Fuzzy Syst 8(6): 746–752
29. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
30. García S, Fernández A, Luengo J, Herrera F (2009) A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Comput 13(10):959–977
31. García S, Herrera F (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res 9: 2579–2596
32. García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heuristics (in press). doi: 10.1007/s10732-008-9080-4
33. Bäck T, Beielstein T (1995) User's group meeting. In: Proceedings of the EuroPVM95: second European PVM, pp 277–282
34. Punch W, Goodman E, Pei M, Chai-shun L, Hovland P, Enbody R (1993) Further research on feature selection and classification using genetic algorithms. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms, pp 557–564
35. Alba E, Dorronsoro B (2008) Cellular genetic algorithms. Springer, Berlin
36. Alba E, Luna F, Nebro A, Troya JM (2004) Parallel heterogeneous genetic algorithms for continuous optimization. Parallel Comput 30(5): 699–719
37. Lin SC, III, WFP, Goodman ED (1994) Coarse-grain parallel genetic algorithms: categorization and new approach. In: Proceedings of the sixth IEEE parallel and distributed processing, pp 28–37
38. Mülhlenbein H, Schomisch M, Born J (1991) The parallel genetic algorithm as function optimizer. Parallel Comput 17(6): 619–632
39. Schlierkamp-Voosen D, Mülhlenbein H (1994) Strategy adaptation by competing subpopulations. In: Parallel solving from nature (PPSN III). Springer, Berlin, pp 199–208
40. Schnecke V, Vornberger O (1996) An adaptative parallel algorithm for vlsi-layout optimization. In: Parallel problem solving from nature (PPSN IV), pp 22–27
41. Tanase R (1989) Distributed genetic algorithms. In: Proceedings of the third international conference on genetic algorithms, pp 434–439
42. Cohoon JP, Hedge S, Martin W (1987) Punctuated equilibria: a parallel genetic algorithm. In: Proceedings of the 2nd international conference on genetic algorithms and their applications, pp 148–154
43. Tanase R (1987) Parallel genetic algorithm for a hypercube. In: Proceedings of the 2nd international conference on genetic algorithms and their applications, pp 177–183

44. Ryan C (1995) Niche and species formation in genetic algorithms. In: Chambers L (ed) Practical handbook of genetic algorithms: applications. CRC Press, Boca Raton, pp 57–74
45. Gürocak HB (1999) A genetic-algorithm-based method for tuning fuzzy logic controllers. Fuzzy Sets Syst 108(1): 39–47
46. Mamdani EH, Assilian S (1975) An experiment in linguistic synthesis with a fuzzy logic controller. Int J Man Mach Stud 7: 1–13
47. Eshelman LJ (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: Rawlin G (ed) Foundations of genetic algorithms, vol 1. Morgan Kaufman, pp 265–283
48. Eshelman L, Schaffer J (1993) Real-coded genetic algorithms and interval-schemata. Found Genet algorithm 2:187–202
49. Kröger B, Schwenderling P, Vornberger O (1993) Parallel genetic packing on transputers. Parallel genetic algorithms: theory and applications, pp 151–186
50. Alcalá-Fdez J, Sánchez L, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera F (2009) KEEL: a software tool to assess evolutionary algorithms to data mining problems. Soft Comput 13(3): 307–318
51. Wang LX, Mendel JM (1992) Generating fuzzy rules by learning from examples. IEEE Trans Syst Man Cybern 22(6): 1414–1427
52. Sheskin D (2003) Handbook of parametric and nonparametric statistical procedures. Chapman & Hall/CRC, Boca Raton
53. Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1:80–83
54. Zar J (1999) Biostatistical analysis. Prentice-Hall, Upper Saddle River