algorithms

Article

Revised Gravitational Search Algorithms Based on Evolutionary-Fuzzy Systems

Danilo Pelusi *, Raffaele Mascella and Luca Tallini

Department of Communication Sciences, University of Teramo, 64100 Teramo, Italy; rmascella@unite.it (R.M.); ltallini@unite.it (L.T.)

* Correspondence: dpelusi@unite.it; Tel.: +39-0861-266036

Academic Editor: Oscar Castillo

Received: 25 January 2017; Accepted: 18 April 2017; Published: 21 April 2017

Abstract: The choice of the best optimization algorithm is a hard issue, and it sometime depends on specific problem. The Gravitational Search Algorithm (GSA) is a search algorithm based on the law of gravity, which states that each particle attracts every other particle with a force called gravitational force. Some revised versions of GSA have been proposed by using intelligent techniques. This work proposes some GSA versions based on fuzzy techniques powered by evolutionary methods, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Differential Evolution (DE), to improve GSA. The designed algorithms tune a suitable parameter of GSA through a fuzzy controller whose membership functions are optimized by GA, PSO and DE. The results show that Fuzzy Gravitational Search Algorithm (FGSA) optimized by DE is optimal for unimodal functions, whereas FGSA optimized through GA is good for multimodal functions.

Keywords: gravitational search algorithm; fuzzy systems; evolutionary algorithm

1. Introduction

Solving optimization problems using exhaustive search techniques is not always the best way. In fact, in the problems with huge dimensional space search, the classical optimization algorithms do not provide a suitable solution. Many researchers take on the problem to optimize objective functions by designing algorithms inspired by the behaviors of natural phenomena. The issue of tuning some parameters of a search algorithm is one of the most important areas of research in evolutionary computation [1]. This research line was followed by Montiel et al. [2] and Castillo et al. [3], which treated the idea of adjusting an evolutionary algorithm.

Among the intelligent evolutionary optimization methods, the Genetic Algorithms (GA) [4] are heuristic approaches well suited to solve complex computational problems [5–9]. Another evolutionary algorithm is the Particle Swarm Optimization (PSO), which depends on the simulation of social behavior [10]. This search method has been applied in various optimization problems [11–15]. An algorithm that operates through the computational steps of a standard evolutionary algorithm is the Differential Evolution (DE). It employs the difference of the parameter vectors to explore the objective function landscape [16]. DE algorithms are applied in many fields [17–21].

Recently, a new search algorithm based on the law of gravity has been proposed: the Gravitational Search Algorithm (GSA) [22]. This algorithm is based on Newton's law of gravity, which states that each particle attracts every other particle with a force called gravitational force [23]. The speed of the search process of GSA depends on parameters that play the main role in the search process. These parameters can be optimized by using evolutionary algorithms such as PSO and DE. However, they may be tuned through fuzzy systems. Askari and Zahiri [24,25] designed fuzzy controllers able to check the search parameters of GSA. In order to improve the performances of GSA, suitable fuzzy logic controllers have been proposed [26–28].

Improvements of the original version of GSA have been achieved by using the binary discrete space. In fact, there are many optimization problems [13,29–34], where the solutions are encoded as binary vectors. Rashedi et al. [35] proposed a binary version of GSA, called BGSA. This work shows the efficiency of BGSA in solving various nonlinear benchmark functions.

This paper aims to improve GSA for certain benchmark functions. The task is to design a fuzzy system able to tune suitable parameters of GSA, taking into account the exploration, the exploitation capabilities and escaping being trapped in local optima. The novelty of the proposed approach is the choice of adjusting, in a fuzzy way, a specific GSA parameter, which may be used both as fuzzy input and output. The fuzzy input is the parameter value in the previous state, whereas the fuzzy output is the GSA parameter value in the current state. In this way, the GSA parameter is tuned depending on the parameter value in the previous state and the remaining fuzzy inputs. Moreover, to assure an intelligent strategy for exploration and exploitation, the fuzzy rules are designed to prevent problems of getting trapped into local optima and premature convergence. The main idea is to design a Fuzzy Gravitational Search Algorithm (FGSA) where the membership functions of the fuzzy controller are optimized by applying evolutionary algorithms such as GA, PSO and DE. The complexity problem is avoided because the fuzzy controller is optimized just one time and than used in GSA. Therefore, FGSA contains the contribution of the optimized fuzzy controller, which does not add considerable complexity with respect to GSA. Finally, the challenge of this work is to improve the average best so far solution of [22,26–28,35] for certain benchmark functions.

The paper is organized as follows. Section 2 describes GSA with the same notation of [22]. The revised GSA optimized through intelligent techniques is described in Section 3. Section 4 contains the discussion of the simulation results of the proposed algorithms. Section 5 concludes the paper.

2. The Gravitational Search Algorithm

The gravitational search algorithm was introduced for the first time by Rashedi et al. in [22]. Their main idea was to consider the searcher agents as a collection of masses that interact with each other based on Newtonian gravity and the laws of motion.

In nature, there are four fundamental interactions [36]: the electromagnetic force, the weak nuclear force, the strong nuclear force and the gravitational force. Newton's law of gravity states that every point mass attracts every single other point mass by a force pointing along the line intersecting both points [23,36]. The gravitational force F is defined by the formula (1) where M_1 and M_2 are the masses of the particles, G is the gravitational constant and R is the distance between the two particles. Note that, the force F is directly proportional to the product of the masses M_1 and M_2 and inversely proportional to the square of the distance between the particles.

$$F = G\frac{M_1 M_2}{R^2} \tag{1}$$

The second law of Newton [23] states that the ratio between a force F applied on a particle of mass M is equal to the acceleration a of the particle (see (2)).

$$a = \frac{F}{M} \tag{2}$$

In order to describe the gravitational search algorithm, the definitions of active gravitational mass, passive gravitational mass and inertial mass are needed. The active gravitational mass is a measure of the strength of the gravitational field due to a particular object. The passive gravitational mass is a measure of the strength of an object's interaction with the gravitational field. The inertial mass is a measure of an object resistance of changing its state of motion when a force is applied.

Let $X_i = (x_i^1, ..., x_i^d, ..., x_i^n,)$ be the position of the i-th agent for $i = 1, 2, ..., n$ where x_i^d represents the position of the i-th agent in the d-th dimension. The force acting on mass i from mass j at a specific time t is denoted with $F_{ij}^d(t)$ (see (3)), where $G(t)$ is the gravitational constant, which depends on time,

M_{aj} is the active gravitational mass related to agent j, M_{pi} is the passive gravitational mass related to agent i, ϵ is a small constant and $R_{ij}(t)$ is the Euclidean distance between two agents i and j as defined in (4).

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{ai}(t)}{R_{ij}(t) + \epsilon}(x_j^d(t) - x_i^d(t)) \tag{3}$$

$$R_{ij}(t) = ||X_i(t), X_j(t)||_2 \tag{4}$$

In the gravitational search algorithm, $G(t)$ is defined as (5) [22], where G_0 is the initial value of G, T is the total number of iterations and α is a parameter.

$$G(t) = G_0 \exp(-\alpha\frac{t}{T}) \tag{5}$$

The total force $F_i^d(t)$ that acts on agent i in a dimension d is a randomly weighted sum of d-th components of the forces exerted from other agents (see Equation (6)). In (6), $rand_j$ is a random number that lies between $[0, 1]$.

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t) \tag{6}$$

By the law (2), it follows that the acceleration of the agent i at time t and in direction d-th, $a_i^d(t)$, is given by (7), where M_{ii} is the inertial mass of the i-th agent.

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} \tag{7}$$

The velocity of an agent at $t+1$ time is a fraction of the velocity at t time added to its acceleration $a(t)$ (see (8) with $rand_i$ uniform random variable in the interval $[0, 1]$). Moreover, the position at $t+1$ time $x_i^d(t+1)$ is calculated through (9).

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{8}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{9}$$

The gravitational and inertial masses are updated by the Equations (10)–(12), where $fit_i(t)$ represents the fitness value of the agent i at time t, and $worst(t)$ and $best(t)$ are defined as in (13) and (14) for minimization (maximization) problems.

$$M_{ai} = M_{pi} = M_{ii} = M_i, i = 1, 2, ..., n \tag{10}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{11}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{n} m_j(t)} \tag{12}$$

$$best(t) = \min_{j\in 1,\dots,n} \left(\max_{j\in 1,\dots,n}\right) fit_j(t) \tag{13}$$

$$worst(t) = \max_{j\in 1,\dots,n} \left(\min_{j\in 1,\dots,n}\right) fit_j(t) \tag{14}$$

In order to find a good compromise between exploration and exploitation, the number of agents with a lapse of time has to be reduced. To improve the performance of GSA by controlling exploration and exploitation, only the K_{best} agents will attract the others [22]. Therefore, Equation (6) can be rewritten as in (15):

$$F_i^d(t) = \sum_{j \in K_{best}, j \neq i}^{n} rand_j F_{ij}^d(t) \tag{15}$$

Algorithm 1 describes the gravitational search algorithm.

Algorithm 1

S1. First of all, the algorithm initializes, in a random way, the position X of the agents in the search space. The initialization procedure depends on the number of agents n, the dimension of the test functions r and the allowable range $[up, down]$ for the search space. The positions X's are calculated by generating $n \times r$ random numbers between 0 and 1 and normalizing them on the range $[up, down]$. In other terms, the positions are computed by the formula $X(i,j) = rand(i,j) * (down - up) + down$, with $i = 1, ..., n$ and $j = 1, ..., r$.
S2. Once the positions are generated, the algorithm computes the fitness of the agents for a certain function. In other words, the value of the objective function is computed. Note that the fitness depends on the position of the agents and the dimension of the function.
S3. The value of the fitness is necessary to calculate the mass of each agent $m_i(t)$ for $i = 1, 2, ..., n$. (see (11)). On the other hand, $m_i(t)$ depends on $best(t)$ and $worst(t)$ (Equations (13) and (14), respectively), which also depend on fitness. Subsequently, the gravitational constant $G(t)$, as defined in (5), is computed.
S4. According to $G(t)$, the total force in different directions is calculated. In particular, the algorithm computes the Euclidean distance between two agents $R_{ij}(t)$ and the force acting on mass i from mass j at a specific time t (see (3)). Subsequently, the total force $F_i^d(t)$ (see (6)) is computed. Therefore, the acceleration of the agents, their velocity and position at t time are calculated by means of (7), (8) and (9), respectively.
S5. The steps from **S2–S4** are repeated until a certain number of iterations is reached.

3. The Evolutionary Fuzzy Algorithms

The first task of our study is the choice of the GSA parameter, which must be tuned. Because the gravitational constant G defined in (5) has a significant effect on the control of the exploration versus exploitation propensities of the particle swarm [25], the parameter α of (5) is chosen. The adjusting of α is accomplished through a fuzzy controller optimized by GA, PSO and DE.

The first step to design a fuzzy controller is the definition of the fuzzy inputs and outputs. The number of fuzzy inputs and outputs depends on the specific problem. In this case, four fuzzy inputs are defined. The first input is the iteration number $N = \{1, 2, ..., k\}$. Let $P_d(i)$ be the population diversity at the i-th iteration, with $i = 1, ..., k$; by considering the (4), the population diversity is defined as in (16), where $R_{mean}(i)$, $R_{min}(i)$, $R_{max}(i)$ are the mean, min and max Euclidean distance between two agents, respectively. The population diversity is the main tool for monitoring the convergence rate of the algorithm, and it represents the second fuzzy input.

$$P_d(i) = \frac{R_{mean}(i) - R_{min}(i)}{R_{max}(i) - R_{min}(i)} \tag{16}$$

Another important parameter for monitoring the convergence rate is the population progress at the i-th iteration denoted by $P_p(i)$. By considering the (13) and (14), the definition of $P_p(i)$ is shown in (17), where $fit_{mean}(i)$ is the fit mean value at the i-th iteration. $P_p(i)$ is the third fuzzy input.

$$P_p(i) = \frac{fit_{mean}(i-1) - fit_{mean}(i)}{fit_{mean}(i)} \tag{17}$$

The fourth fuzzy input $\alpha(i-1)$ is the value of α at the $(i-1)th$ iteration. Because the parameter α has to be tuned, the fuzzy output is the value $\alpha(i)$, i.e., the value of α at the i-th iteration.

The next step is the choice of the number and shape of the membership functions. This choice depends on the specific problem, and sometimes, it is a trial and error procedure. By considering our problem, triangular/trapezoidal membership functions are chosen. The inputs N, $\alpha(t-1)$ and the output $\alpha(t)$ have three membership functions: Low (L), Medium (M) and High (H). The other inputs have two membership functions: Low (L) and High (H). The ranges of α are defined experimentally for each benchmark function, whereas the ranges of N, P_d and P_p are the same for all of the test functions.

The number of membership functions for each input defines the fuzzy rules number. Therefore, our rule set is composed of $3 \times 2 \times 2 \times 3 = 36$ rules (see Table 1).

Table 1. Fuzzy rules.

Rule	**Fuzzy Inputs**				**Fuzzy Output**
R	N	P_d	P_p	$\alpha(i-1)$	$\alpha(i)$
1	L	L	L	L	L
2	L	L	L	M	L
3	L	L	L	H	L
4	L	L	H	L	L
5	L	L	H	M	L
6	L	L	H	H	M
7	L	H	L	L	L
8	L	H	L	M	L
9	L	H	L	H	M
10	L	H	H	L	L
11	L	H	H	M	M
12	L	H	H	H	H
13	M	L	L	L	L
14	M	L	L	M	L
15	M	L	L	H	M
16	M	L	H	L	L
17	M	L	H	M	L
18	M	L	H	H	M
19	M	H	L	L	L
20	M	H	L	M	L
21	M	H	L	H	M
22	M	H	H	L	L
23	M	H	H	M	M
24	M	H	H	H	H
25	H	L	L	L	L
26	H	L	L	M	M
27	H	L	L	H	H
28	H	L	H	L	M
29	H	L	H	M	H
30	H	L	H	H	H
31	H	H	L	L	M
32	H	H	L	M	H
33	H	H	L	H	H
34	H	H	H	L	H
35	H	H	H	M	H
36	H	H	H	H	H

The rules shown in Table 1 are designed to prevent problems of being trapped in local optima and premature convergence. This fact justifies the dependance of the (16), (17) and α from the number of iterations; in other terms, $P_d = P_d(i)$, $P_p = P_p(i)$ and $\alpha = \alpha(i)$ with $i = 1, ..., k$.

All of the rules have been generated through knowledge. The presence of a lack of improvement when the number of iterations is low is a sign of being trapped in the local optimum and premature convergence. If the number of iterations is low and $\alpha(i-1)$ is low, then $\alpha(i)$ is set to low. In fact, when α is low, the value of the gravitational constant G increases. Therefore, the acceleration and velocity of the agents increase, and the population can escape from being trapped in local optima. In the situations where the population diversity is low and the number of iterations is medium, a reduction of α improves the diversity to achieve better solutions.

In order to increase the power of exploitation of the algorithm, when the iteration number is high, the gravitational constant must decrease. Therefore, by increasing α, it follows that G decreases.

If the algorithm is at the last iterations, there is a lack of improvement and the diversity is huge, the algorithm has failed the convergence target. In these situations, α must be increased. Finally, all of the rules follow the knowledge described before.

Our fuzzy system uses the Mamdani inference system, and the inputs are combined logically using the AND operator. The centroid is the used defuzzification method.

In the evolutionary fuzzy systems, the membership functions' shapes can be evolved using a genetic algorithm [37]. Setnes et al. [38] proposed a real-coded GA, which simultaneously optimizes the parameters of the antecedent membership functions and the rule consequent. By following this way, we apply a suitable genetic algorithm to establish the best slopes of the triangular/trapezoidal membership functions.

In order to achieve a good optimization, the optimization range for all of the membership functions has to be defined. The ranges are established avoiding possible crossings of more than two membership functions. The inputs N, P_d and P_p have the same ranges for all of the simulations. The whole range of the iteration number N is $[0, 1000]$ (i.e., $k = 1000$), whereas the population diversity P_d and population progress P_p are normalized in $[0, 1]$. On the contrary, the optimization intervals of the α membership functions are determined empirically for each benchmark function.

In this work, we consider real-coded GAs [39] because binary coded or classical GAs [40] are less efficient when applied to multidimensional, high-precision or continuous problems. In fact, the bit strings can become very long, and the search space blows up.

A serious problem of the genetic algorithms' design is the definition of the fitness function [41]. Bad fitness functions can easily make the algorithm get trapped in a local optimum solution and lose the discovery power. Petridis et al. [42] presented a specific varying fitness function technique that incorporates the problem's constraints into the fitness function in a dynamic way. However, our evaluation of the fitness functions is based on two features: efficiency and precision. Good fitness functions help the algorithm to explore the search space more effectively and efficiently. In order to assure such features, the fitness function $f(x)$ shown in (18) is defined.

$$f(x) = \frac{1}{1 + exp(\sqrt[4]{x})}; \tag{18}$$

In Equation (18), the variable x represents the best so far solution [22] to the iteration number $k = 1000$. The genetic algorithm searches the value of x that maximizes the fitness Function (18). Such a value of x will be as small as possible. The slope of the membership functions is adjusted until the optimal value of (18) is achieved.

The genetic algorithms are characterized by three genetic procedures: selection, crossover and mutation. Among the various selection methods, we choose the roulette wheel selection method [43].

In roulette wheel selection, the probability that individual i is selected, $P(I = i)$, is computed by Equation (19), where l is the population number.

$$P(I = i) = \frac{f_i(x)}{\sum_{j=1}^{l} f_j(x)}; \tag{19}$$

Algorithm 2 implements the roulette wheel selection.

Algorithm 2

```
S1. For each individual $i$, with $i = 1, 2, ..., l$:
  S1.1. Generate a random number $r$ between 0 and 1.
  S1.2. Initialize the variables $sum$ to 0 and $j$ to 1.
  S1.3. If $sum \geq r$ go to S.1.4.
    S1.3.1. Compute $sum = sum + P(I = i)$, where $P(I = i)$ is defined in (19).
    S1.3.2. Increment $j$.
    S1.3.3. Go to Step S1.3.
  S1.4. Set $parent(i) = par(j - 1)$, i.e., establish the $i$-th parent $parent$ from the $(j - 1)$-th value of the parameter $par$.
S2. Print the values of the parents.
```

In order to improve the convergence of the GA, we consider the elitism procedure [44]. It is an addition to many selection methods that forces the GA to retain some number of the best individuals at each generation. Some studies [45,46] show that elitism can increase the performance of GA by preventing the loss of good solutions once they are found.

Another genetic procedure is the crossover. The crossover recombines genetic material of parent chromosomes to produce offspring on the basis of crossover probability [47]. Given the mutations number $nmutation$, the number of elitism $nelit$, the number of the population l and the parents $parent(i)$ $(i = 1, ..., l)$, the designed crossover algorithm follows the steps of Algorithm 3.

Algorithm 3

```
S1. For all $j$ from 1 to $\left\lfloor \frac{l - nmutation - nelit}{2} \right\rfloor$:
  S1.1. Compute a random number $t$ between $-0.25$ and 1.25
  S1.2. Compute the parameters $p(2j - 1) = t \cdot parent(2j - 1) + (1 - t) \cdot parent(2j)$ and $p(2j) = t \cdot parent(2j) + (1 - t) \cdot parent(2j - 1)$.
S2. Print the parameters $p$
```

The mutation is a genetic operation that occasionally breaks one or more members of a population out of a local minimum/maximum space and potentially discovers a better minimum/maximum space [44]. Algorithm 4 describes the mutation algorithm, where the mutations number $nmutation$ and the random mutations number $nmutationR$ [41,43] are given. Moreover, the variable $sigma$ in Algorithm 4 is calculated through the formula (20) where $max(p_r)$ and $min(p_r)$ are the max and min value of the initial random parameters p_r.

$$sigma = \frac{max(p_r) - min(p_r)}{10}; \tag{20}$$

In our algorithm, the number l of population P is set to 100; the number of mutation m is equal to 20; and number of elitism is equal to two. The designed fuzzy controller is optimized through GA: the steps of the optimization algorithm are described in the Algorithm 5. GA optimizes the slope of the membership functions (four parameters for each membership function), to achieve the optimal membership functions. Because the membership functions number is 13, it follows that $13 \times 4 = 52$ parameters are optimized. Once the fuzzy controller is optimized, it may be used to adjust the parameter α of GSA. The fuzzy gravitational search algorithm optimized by GA is described in Algorithm 6.

Algorithm 4

```
**S1**. For all the $i$ from $l - nmutation + 1$ to $l - nmutationR$:
**S1.1**. Compute a random number $phi$ between $-1$ and $+1$.
**S1.2**. Compute $z = erfinv(phi) * (2^{0.5})$, where $erfinv$ is the inverse error function.
**S1.3**. Compute the parameter $p$ with the formula $p(i) = z \cdot (sigma) + parent(i)$, where $sigma$ is calculated by (20).
**S2**. Print the parameters $p$.
```

Algorithm 5

```
**S1**. Pass the optimization ranges of the membership functions to GA.
**S2**. Select in a random way the initial optimal values of the parameters in the optimization ranges and calculate the termination criteria of GA. This criteria depends on number of generations and fitness function values.
**S3**. If the termination criteria is achieved, go to **S10**.
**S4**. For each $j$-th element of the population $P$, with $j = 1, ..., l$, compute the steps from **S4.1–S4.4**
**S4.1**. Pass the optimal parameters to the fuzzy inputs/output of the controller.
**S4.2**. Initialize, in a random way, the position $X$ of the agents in the search space. The initialization procedure depends on the number of agents $n$, the dimension of the test functions $r$ and the allowable range $[up, down]$ for the search space. The positions $X$'s are calculated by generating $n \times r$ random numbers between 0 and 1, and normalize them on the range $[up, down]$. In other terms, the positions are computed by the formula $X(i,j) = rand(i,j) * (down - up) + down$, with $i = 1, ..., n$ and $j = 1, ..., r$.
**S4.3**. Set the initial value $\alpha(1)$ and for each iteration $i$, with $i = 1, 2, ..., k$, compute the steps from **S4.3.1–S4.3.4**
**S4.3.1**. Compute the fitness of the agents for a certain function. The fitness depends on the position of the agents and the dimension of the function.
**S4.3.2**. The value of the fitness is necessary to calculate the mass of each agent $m_i(t)$ for $i = 1, 2, ..., n$. (see (11)). On the other hand, $m_i(t)$ depends on $best(t)$ and $worst(t)$ (Equations (13) and (14), respectively), which also depend on fitness. The gravitational constant $G(t)$ is computed according to $\alpha(1)$ at the first iteration, whereas, for the other iterations, $G(t)$ is calculated according to the tuned value of $\alpha$.
**S4.3.3**. According to $G(t)$, the total force in different directions is calculated. In particular, the algorithm computes the Euclidean distance between two agents $R_{ij}(t)$ and the force acting on mass $i$ from mass $j$ at a specific time $t$ (see (3)). Subsequently, the total force $F_i^d(t)$ (see (6)) is computed. Therefore, the acceleration of the agents, their velocity and position at $t$ time are calculated by means of (7), (8) and (9), respectively.
**S4.3.4**. Compute the fuzzy output $\alpha(i+1)$, i.e., the $i+1$ value of $\alpha$, through the fuzzy controller, which receives the following inputs: iteration number $i$, population diversity $P_d(i)$, population progress $P_p(i)$ and $\alpha(i)$.
**S4.4**. Calculate the $j$-th value of the fitness function $f_j(x)$ defined in (18).
**S5**. Update the termination criteria parameters, which depend on fitness function values.
**S6**. Compute the parents $parent(i)$ with $i = 1, ..., l$ through the selection method by using the Algorithm 2.
**S7**. Compute the parameters $p$ through the crossover method described in the Algorithm 3.
**S8**. Compute some parameters $p$ with the mutation (Algorithm 4).
**S9**. Go to Step **S3**.
**S10**. Give to the output the optimal parameters of the membership functions.
```

In order to test the algorithms, unimodal and multimodal benchmark functions [48] are considered. Table 2 shows the test functions, where r is the dimension of the function and S is a subset of R^r.

The membership functions of the fuzzy controller can be optimized also through evolutionary algorithms such as PSO and DE. The idea is to optimize the parameter α of GSA by PSO and DE. In this case, only the ranges of the α membership functions are optimized. Therefore, 3×4 (alpha previous state) + 3×4 (alpha current state) = 24 parameters are optimized. In the PSO and DE case, the membership functions ranges of α are defined considering a neighborhood of the optimal value of α that come from PSO [10] (DE [16]) optimization. The steps of FGSA-PSO (DE) algorithm are illustrated in Algorithm 7.

Algorithm 6

S1. Initialize, in a random way, the position X of the agents in the search space. The initialization procedure depends on the number of agents n, the dimension of the test functions r and the allowable range $[up, down]$ for the search space. The positions X's are calculated by generating $n \times r$ random numbers between 0 and 1, and normalize them on the range $[up, down]$. In other terms, the positions are computed by the formula $X(i, j) = rand(i, j) * (down - up) + down$, with $i = 1, ..., n$ and $j = 1, ..., r$.
S2. Set the initial value $\alpha(1)$ and for each iteration i, with $i = 1, 2, ..., k$, compute the steps from **S2.1–S2.5**.
S2.1. Compute the fitness of the agents for a certain function. The fitness depends on the position of the agents and the dimension of the function.
S2.2. The value of the fitness is necessary to calculate the mass of each agent $m_i(t)$ for $i = 1, 2, ..., n$. (see (11)). On the other hand, $m_i(t)$ depends on $best(t)$ and $worst(t)$ (Equations (13) and (14), respectively), which also depend on fitness. The gravitational constant $G(t)$ is computed according to $\alpha(1)$ at the first iteration, whereas, for the other iterations, $G(t)$ is calculated according to the adjusted value of α (which comes from fuzzy controller).
S2.3. According to $G(t)$, the total force in different directions is calculated. In particular, the algorithm computes the Euclidean distance between two agents $R_{ij}(t)$ and the force acting on mass i from mass j at a specific time t (see (3)). Subsequently, the total force $F_i^d(t)$ (see (6)) is computed. Therefore, the acceleration of the agents, their velocity and position at t time, are calculated by means of (7), (8) and (9), respectively.
S2.4. Calculate the population diversity $P_d(i)$ and the population progress $P_p(i)$ as defined in (16) and (17).
S2.5. The GA-optimized fuzzy controller receives the inputs: iteration number i, population diversity $P_d(i)$, population progress $P_p(i)$ and $\alpha(i)$. The controller gives the fuzzy output $\alpha(i+1)$, i.e., the $i+1$ value of α. In this way, the value of α is tuned.
S3. Give to the output the best-so-far solution.

Table 2. Test functions.

Unimodal Functions	***S***
$F_1(X) = \sum_{i=1}^{r} x_i^2$	$[-100; 100]^r$
$F_2(X) = \sum_{i=1}^{r} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10; 10]^r$
$F_3(X) = \sum_{i=1}^{r} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100; 100]^r$
$F_4(X) = max_i\{\lvert x_i \rvert, 1 \le i \le r\}$	$[-100; 100]^r$
$F_5(X) = \sum_{i=1}^{r-1} \left[100\,(x_{i+1} - x_i)^2 + (x_i - 1)^2 \right]$	$[-30; 30]^r$
$F_6(X) = \sum_{i=1}^{r} ([x_i + 0.5])^2$	$[-100; 100]^r$
$F_7(X) = \sum_{i=1}^{r} i \cdot x_i^4 + random[0, 1)$	$[-1.28; 1.28]^r$
Multimodal Functions	***S***
$F_8(X) = \sum_{i=1}^{r} -x_i sin\sqrt{\lvert x_i \rvert}$	$[-500; 500]^r$
$F_9(X) = \sum_{i=1}^{r} \left[x_i^2 - 10cos(2\pi x_i + 10) \right]$	$[-5.12; 5.12]^r$
$F_{10}(X) = -20exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{r} x_i^2} \right) +$ $-exp\left(\frac{1}{r}\sum_{i=1}^{n} cos(2\pi x_i) \right) + 20 + e$	$[-32; 32]^r$
$F_{11}(X) = \frac{1}{4000}\sum_{i=1}^{r} x_i^2 - \prod_{i=1}^{r} cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$	$[-600; 600]^r$
$F_{12}(X) = \frac{\pi}{r}\{10sin(\pi y_1) + \sum_{i=1}^{r-1}(y_i - 1)^2[1 + 10sin^2(\pi y_{i+1})] +$ $(y_r - 1)^2\} + \sum_{i=1}^{r} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & a < x_i < a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	$[-50; 50]^r$
$F_{13}(X) = 0.1\{sin^2(3\pi x_1) + \sum_{i=1}^{r}(x_1 - 1)^2[1 + sin^2(3\pi x_1 + 1)]$ $+(x_r - 1)^2[1 + sin^2(2\pi x_r)]\} + \sum_{i=1}^{r} u(x_i, 5, 100, 4)$	$[-50; 50]^r$

Algorithm 7

S1. Optimize GSA with PSO (DE); in this way, the optimal value α_{opt} is computed.
S2. Calculate a suitable neighborhood of α_{opt} with the formulas: $\alpha_{opt_{inf}} = \alpha_{opt} - \epsilon$ and $\alpha_{opt_{sup}} = \alpha_{opt} + \epsilon$, with $\epsilon = 0.02$.
S3. Compute the membership functions ranges of α taking into account the neighborhood of α_{opt}.
S4. Initialize, in a random way, the position X of the agents in the search space. The initialization procedure depends on the number of agents n, the dimension of the test functions r and the allowable range $[up, down]$ for the search space. The positions X's are calculated by generating $n \times r$ random numbers between 0 and 1, and normalize them on the range $[up, down]$. In other terms, the positions are computed by the formula $X(i,j) = rand(i,j) * (down - up) + down$, with $i = 1, ..., n$ and $j = 1, ..., r$.
S5. Set the initial value $\alpha(1)$ and for each iteration i, with $i = 1, 2, ..., k$, compute the steps from **S5.1–S5.5**.
S5.1. Compute the fitness of the agents for a certain function. The fitness depends on the position of the agents and the dimension of the function.
S5.2. The value of the fitness is necessary to calculate the mass of each agent $m_i(t)$ for $i = 1, 2, ..., n$. (see (11)). On the other hand, $m_i(t)$ depends on $best(t)$ and $worst(t)$ (Equations (13) and (14), respectively), which also depend on fitness. The gravitational constant $G(t)$ is computed according to $\alpha(1)$ at the first iteration, whereas, for the other iterations, $G(t)$ is calculated according to the adjusted value of α (which comes from fuzzy controller).
S5.3. According to $G(t)$, the total force in different directions is calculated. In particular, the algorithm computes the Euclidean distance between two agents $R_{ij}(t)$ and the force acting on mass i from mass j at a specific time t (see (3)). Subsequently, the total force $F_i^d(t)$ (see (6)) is computed. Therefore, the acceleration of the agents, their velocity and position at t time, are calculated by means of (7), (8) and (9), respectively.
S5.4. Calculate the population diversity $P_d(i)$ and the population progress $P_p(i)$ as defined in (16) and (17).
S5.5. The PSO (DE)-optimized fuzzy controller receives the inputs: iteration number i, population diversity $P_d(i)$, population progress $P_p(i)$ and $\alpha(i)$. The controller gives the fuzzy output $\alpha(i+1)$, i.e. the $i+1$ value of α. In this way, the value of α is tuned.
S6. Give to the output the best-so-far solution.

4. Simulation and Discussion

The proposed algorithms are tested with MATLAB software on a 2.20-GHz CPU hardware for the benchmark functions of Table 2. We firstly apply our algorithms to the unimodal functions, i.e., to the test functions from F_1 to F_7. Subsequently, we take into account the multimodal test functions from F_8 to F_{13}. In all of the cases the number n of agents is set to 50; the dimension of the functions r is 30; the maximum iteration k is 1000.

Algorithms 2–5 give the optimal parameters to define the membership functions. For each benchmark function, different membership functions' slopes are obtained. As an example, Figure 1 shows the optimal membership functions for the test function $F10$.

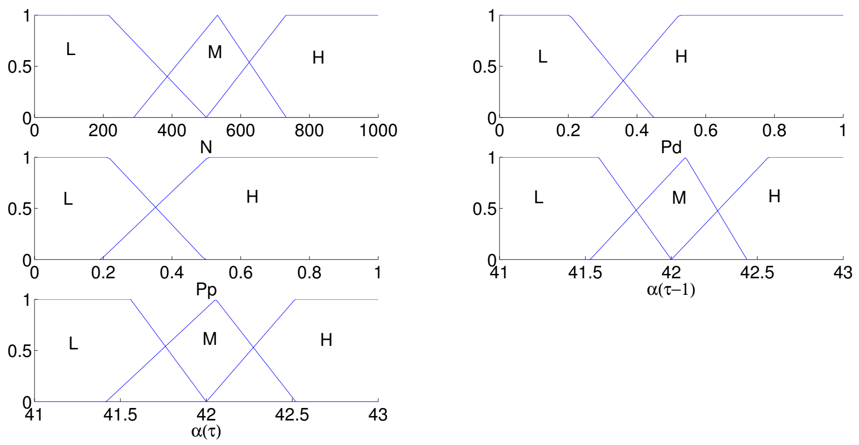

Figure 1. Optimal membership functions for the benchmark function $F10$.

By running Algorithm 6 with the best membership functions, we obtain the solid curve of Figure 2 for the benchmark function $F1$. The dashed line represents the best so far solution of GSA [22].

Note that FGSA-GA tends to find the global optimum faster than GSA, and hence, it has a higher convergence rate.

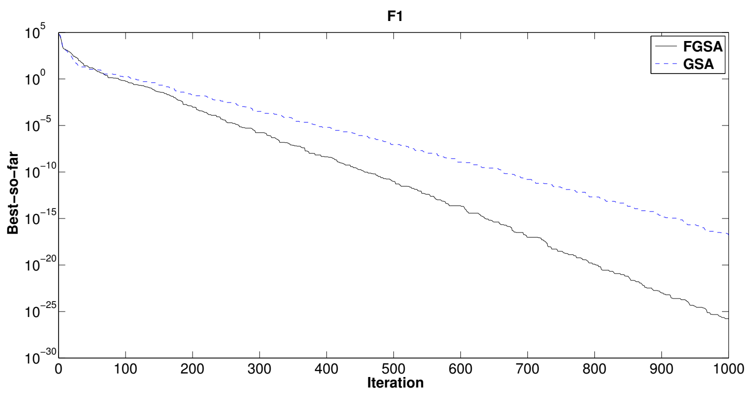

Figure 2. Comparison of the performance of the Gravitational Search Algorithm (GSA) and Fuzzy Gravitational Search Algorithm (FGSA)-GA for minimization of $F1$.

The best so far solutions of GSA and FGSA-GA for $F2$ are shown in Figure 3. Observing Figure 3, we note that up to the 30th iteration, the trend of GSA and FGSA-GA is about the same. After this value, FGSA-GA is better than GSA. Note that the best so far solution achieved at the last iteration is 4.2×10^{-13}.

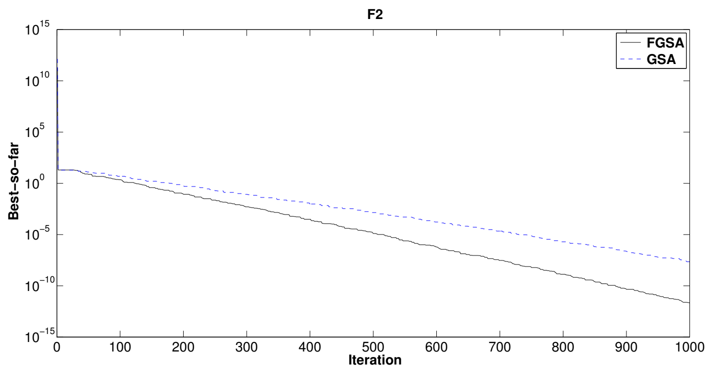

Figure 3. Comparison of the performance of GSA and FGSA-GA for minimization of $F2$.

The trend of the GSA and FGSA-GA best so far solution for $F3$ is shown in Figure 4. The improvement of FGSA-GA with respect to GSA is about four orders of magnitude.

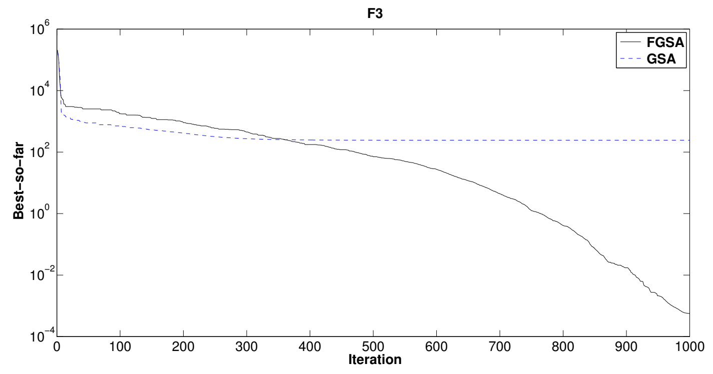

Figure 4. Comparison of the performance of GSA and FGSA-GA for minimization of $F3$.

From Figure 5, we observe that the trend of FGSA-GA is close to GSA: a relevant improvement is achieved at the end of the iterations. Quantitatively, for the function $F4$, there is an improvement of two orders of magnitude.

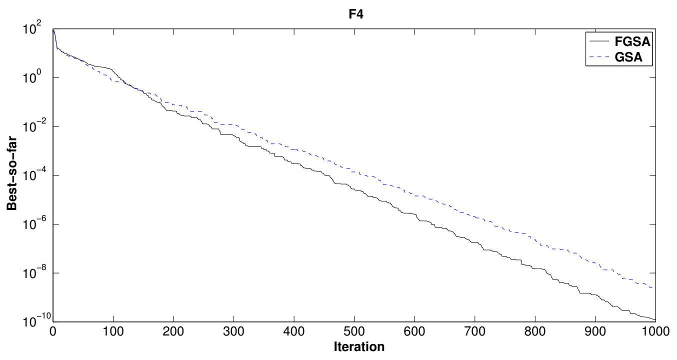

Figure 5. Comparison of the performance of GSA and FGSA-GA for minimization of $F4$.

The multimodal functions are most difficult to optimize because they have many local minima. For multimodal functions, the final results are more important, since they reflect the ability of the algorithm to escape from poor local optima and locating a near-global optimum. Using Algorithm 6, significant improvements are achieved with the test functions $F10$, $F11$ and $F12$.

For the multimodal function $F10$, FGSA-GA is better than GSA from the beginning to the end of the iterations (see Figure 6). In particular, the maximum difference is of 10 orders of magnitude.

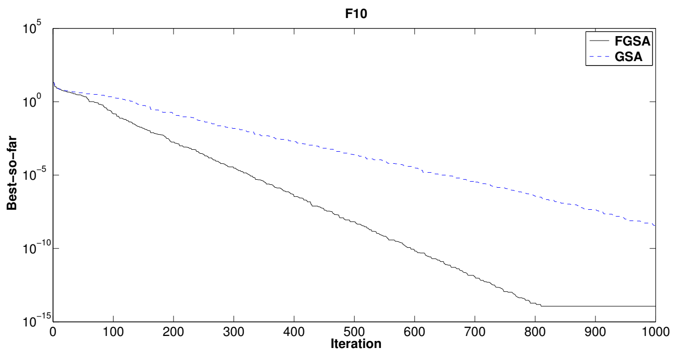

Figure 6. Comparison of the performance of GSA and FGSA-GA for minimization of $F10$.

Observing Figure 7, we can note that FGSA-GA improves GSA by about eight orders of magnitude: FGSA-GA's best so far solution is 9.61×10^{-9}.

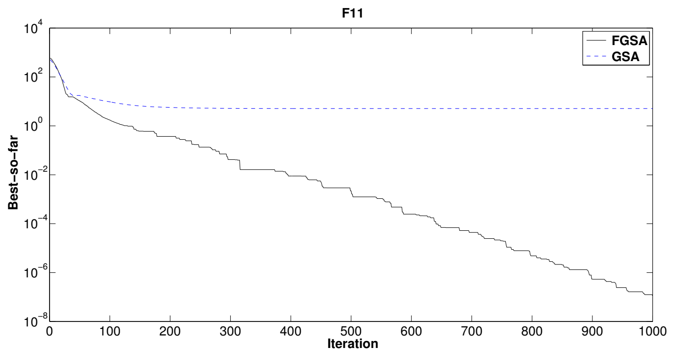

Figure 7. Comparison of the performance of GSA and FGSA-GA for minimization of $F11$.

Figure 8 shows the GSA and FGSA-GA trend over iteration number for function $F12$. Note that the trend is about the same up to the 500th iteration. However, there is an improvement of FGSA-GA at the end of the iterations.

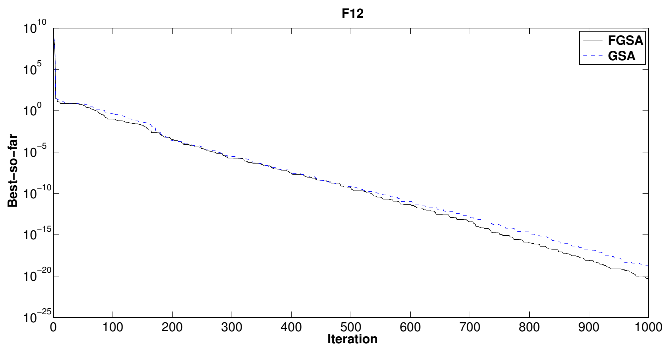

Figure 8. Comparison of the performance of GSA and FGSA-GA for minimization of $F12$.

Algorithm 5 uses GA to optimize the membership functions of the fuzzy system. However, the parameter α of GSA can be optimized with the help of other optimization methods such as PSO and DE. In FGSA, the idea is to define the membership functions range of α, taking into account the optimal value of α obtained with PSO and DE. The procedure is described in Algorithm 7.

We compare the results of GSA and FGSA optimized by PSO (GSA-PSO and FGSA-PSO, respectively). By observing Figure 9, we can note that the best-so-far solution of FGSA-PSO is better than GSA-PSO for the test function $F1$. In fact, from Iteration 700–1000, the FGSA curve is below the GSA curve. There is a good improvement for the test function $F7$ (see Figure 10) where FGSA-PSO overcomes the performances of GSA-PSO. For the other test functions, the results are about the sames.

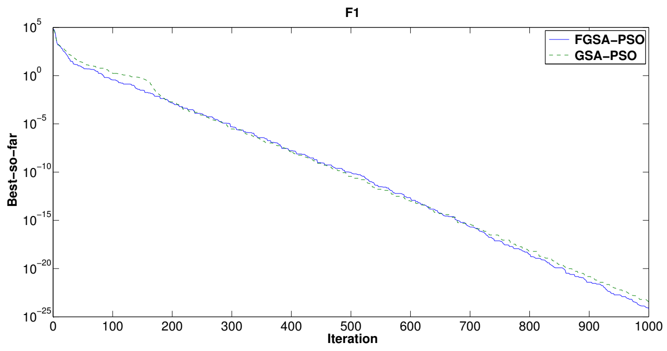

Figure 9. Comparison of performance of FGSA-PSO and GSA-PSO for minimization of $F1$.

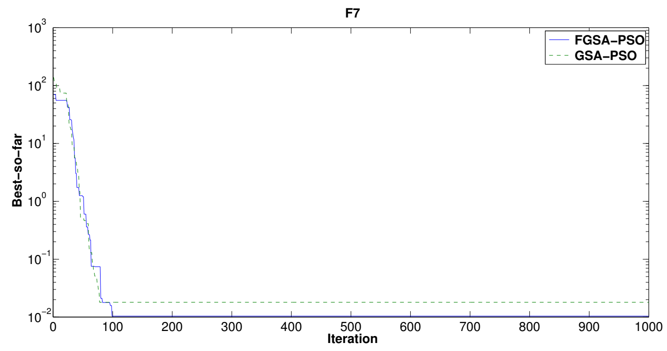

Figure 10. Comparison of the performance of FGSA-PSO and GSA-PSO for minimization of $F7$.

Better results are achieved by FGSA optimized by DE (FGSA-DE) versus GSA optimized by DE (GSA-DE). The results of FGSA-DE come from Algorithm 7. Figure 11 shows the comparison between FGSA-DE and GSA-DE for the minimization of *F*4. Note that the FGSA-DE trend is better than GSA-DE because the FGSA-DE curve is below GSA-DE. A similar situation for the function *F*9 is obtained (see Figure 12). Moreover, for the test function *F*11, there is an improvement of about one order of magnitude (see Figure 13). The simulation results for the function *F*13 show that the FGSA-DE curve is always below GSA-DE (see Figure 14); therefore, FGSA-DE tends to find the global optimum faster than GSA-DE.

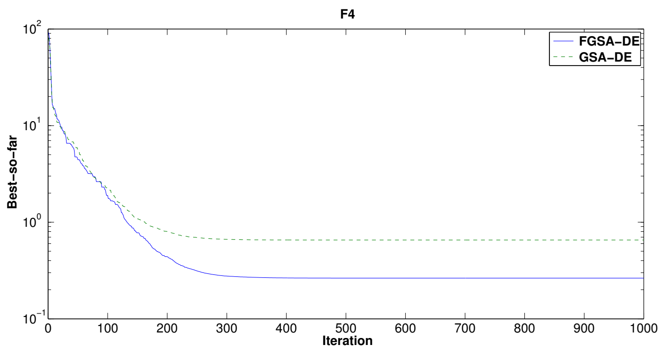

Figure 11. Comparison of the performance of FGSA-Differential Evolution (DE) and GSA-DE for minimization of *F*4.

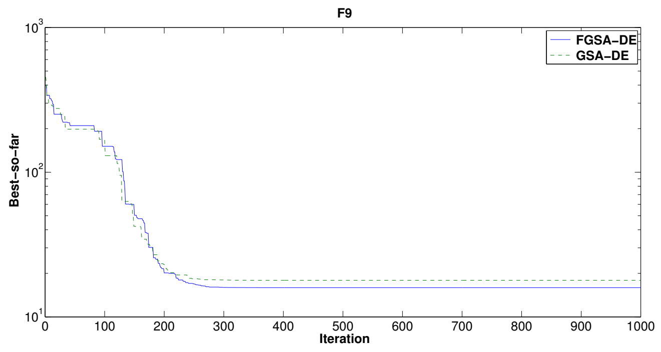

Figure 12. Comparison of the performance of FGSA-DE and GSA-DE for minimization of *F*9.

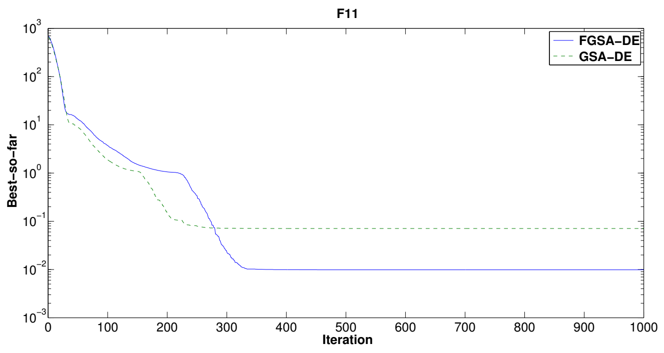

Figure 13. Comparison of the performance of FGSA-DE and GSA-DE for minimization of *F*11.

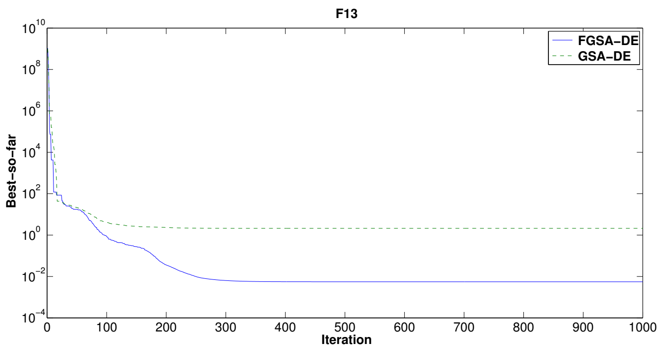

Figure 14. Comparison of the performance of FGSA-DE and GSA-DE for minimization of $F13$.

The results are averaged over 30 and 16 runs, as in [22,26,27,35] and [28], respectively, and the average, standard deviation and median of the best solution in the last iteration are computed.

Tables 3–6 resume the comparison between the algorithms FGSA-GA, FGSA-PSO and FGSA-DE over 30 and 16 runs. The values shown in the tables are the average, standard deviation and median of the best solution in the last iteration. Note that FGSA-DE is better than FGSA-GA and FGSA-PSO for the functions $F1$ and $F2$. On the other hand, FGSA-GA overcomes FGSA-PSO and FGSA-DE for the functions $F3$, $F11$ and $F12$. For the test functions $F5$–$F9$, the results are similar: there is not a dominant algorithm.

Table 3. Comparisons between FGSA-GA, FGSA-PSO and FGSA-DE over 30 runs for unimodal functions.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
FGSA-GA	1.78×10^{-27}	2.32×10^{-13}	3.27×10^{-5}	2.28×10^{-11}	2.11×10^{1}	2.00×10^{-11}	1.76×10^{-2}
	5.61×10^{-27}	4.83×10^{-13}	3.24×10^{-4}	4.27×10^{-12}	3.85×10^{-1}	7.61×10^{-12}	6.32×10^{-3}
	1.62×10^{-26}	2.30×10^{-12}	2.04×10^{-4}	1.26×10^{-10}	2.11×10^{1}	3.20×10^{-11}	1.68×10^{-2}
FGSA-PSO	7.93×10^{-25}	6.16×10^{-7}	5.55×10^{2}	2.85×10^{0}	2.84×10^{1}	4.56×10^{-11}	3.81×10^{-2}
	2.30×10^{-25}	3.49×10^{-7}	2.27×10^{2}	1.57×10^{0}	4.16×10^{1}	7.81×10^{-12}	2.28×10^{-2}
	7.82×10^{-25}	8.05×10^{-7}	5.20×10^{2}	2.57×10^{0}	2.76×10^{1}	5.26×10^{-11}	3.46×10^{-2}
FGSA-DE	2.85×10^{-35}	4.44×10^{-17}	9.68×10^{-3}	3.86×10^{-11}	2.10×10^{1}	8.04×10^{-11}	2.27×10^{-2}
	1.56×10^{-36}	2.35×10^{-18}	3.08×10^{-3}	6.35×10^{-12}	3.08×10^{-1}	6.79×10^{-12}	6.92×10^{-3}
	1.46×10^{-35}	2.95×10^{-17}	1.48×10^{-4}	2.85×10^{-11}	2.10×10^{1}	9.78×10^{-11}	2.16×10^{-2}

Table 4. Comparisons between FGSA-GA, FGSA-PSO and FGSA-DE over 30 runs for multimodal functions.

	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
FGSA-GA	2.65×10^{3}	1.47×10^{1}	7.88×10^{-15}	1.30×10^{-9}	2.44×10^{-21}	2.74×10^{-2}
	3.54×10^{2}	3.28×10^{0}	1.14×10^{-16}	2.69×10^{-10}	5.26×10^{-22}	3.41×10^{-3}
	2.69×10^{3}	1.39×10^{1}	7.99×10^{-15}	2.97×10^{-9}	5.96×10^{-21}	1.59×10^{-2}
FGSA-PSO	2.73×10^{3}	1.77×10^{1}	7.05×10^{-11}	1.35×10^{1}	1.12×10^{-2}	2.08×10^{1}
	4.20×10^{2}	3.96×10^{0}	2.01×10^{-11}	3.87×10^{0}	3.06×10^{-3}	9.56×10^{0}
	2.73×10^{3}	1.79×10^{1}	7.51×10^{-11}	1.27×10	1.13×10^{-2}	1.83×10^{1}
FGSA-DE	2.79×10^{3}	1.46×10^{1}	7.99×10^{-15}	2.89×10^{-2}	1.97×10^{-1}	1.85×10^{-2}
	5.55×10^{2}	3.92×10^{0}	9.33×10^{-16}	4.16×10^{-2}	3.18×10^{-2}	3.03×10^{-3}
	2.71×10^{3}	1.49×10^{1}	7.99×10^{-15}	1.11×10^{-2}	7.55×10^{-2}	2.03×10^{-2}

Table 5. Comparisons between FGSA-GA, FGSA-PSO and FGSA-DE over 16 runs for unimodal functions.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
FGSA-GA	1.73×10^{-26}	2.01×10^{-12}	3.22×10^{-4}	8.29×10^{-11}	2.10×10^{1}	2.50×10^{-11}	2.15×10^{-2}
	3.31×10^{-27}	6.75×10^{-13}	2.99×10^{-5}	2.77×10^{-12}	3.55×10^{-1}	4.47×10^{-12}	1.40×10^{-3}
	1.66×10^{-26}	2.08×10^{-12}	1.90×10^{-4}	1.29×10^{-10}	2.12×10^{1}	2.78×10^{-11}	1.84×10^{-2}
FGSA-PSO	8.91×10^{-25}	7.26×10^{-7}	5.77×10^{2}	2.26×10^{0}	5.07×10^{1}	8.36×10^{-11}	3.38×10^{-2}
	3.27×10^{-26}	2.19×10^{-7}	1.44×10^{2}	1.49×10^{0}	5.01×10^{0}	1.45×10^{-12}	1.54×10^{-2}
	8.28×10^{-25}	7.29×10^{-7}	5.37×10^{2}	2.16×10^{0}	2.76×10^{1}	7.56×10^{-11}	3.18×10^{-2}
FGSA-DE	2.58×10^{-35}	5.17×10^{-17}	2.58×10^{-4}	3.11×10^{-11}	2.12×10^{1}	8.36×10^{-11}	2.41×10^{-2}
	2.23×10^{-35}	1.28×10^{-18}	2.48×10^{-4}	5.78×10^{-12}	1.75×10^{-1}	4.21×10^{-12}	8.14×10^{-3}
	2.00×10^{-35}	3.05×10^{-17}	1.63×10^{-4}	1.98×10^{-11}	2.12×10^{1}	7.78×10^{-11}	2.55×10^{-2}

Table 6. Comparisons between FGSA-GA, FGSA-PSO and FGSA-DE over 16 runs for multimodal functions.

	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
FGSA-GA	2.63×10^{3}	1.56×10^{1}	8.44×10^{-15}	1.70×10^{-9}	2.04×10^{-21}	1.47×10^{-2}
	4.45×10^{2}	3.52×10^{0}	1.21×10^{-15}	4.48×10^{-10}	4.15×10^{-22}	2.90×10^{-3}
	2.49×10^{3}	1.59×10^{1}	7.99×10^{-15}	3.53×10^{-8}	6.73×10^{-21}	3.39×10^{-2}
FGSA-PSO	2.79×10^{3}	1.82×10^{1}	6.44×10^{-11}	1.50×10^{1}	1.07×10^{-2}	1.80×10^{1}
	5.15×10^{2}	4.62×10^{0}	2.78×10^{-11}	3.94×10^{0}	3.26×10^{-3}	1.03×10^{1}
	2.63×10^{3}	1.89×10^{1}	7.38×10^{-11}	1.36×10^{1}	9.66×10^{-3}	1.71×10^{1}
FGSA-DE	2.65×10^{3}	1.55×10^{1}	7.99×10^{-15}	6.18×10^{-2}	2.68×10^{-1}	2.23×10^{-2}
	3.57×10^{2}	4.30×10^{0}	1.30×10^{-15}	6.37×10^{-2}	3.69×10^{-1}	3.21×10^{-3}
	2.66×10^{3}	1.49×10^{1}	7.99×10^{-15}	5.37×10^{-2}	1.13×10^{-1}	1.38×10^{-2}

Table 7 shows the comparison of the results achieved by [22,26,27,35] with the performances of the algorithms FGSA-GA, FGSA-PSO and FGSA-DE over 30 runs. This table does not contain the results of all 13 test functions, but it resumes the more significant improvements for certain benchmark functions. The last three columns of Table 7 contain the average of the best solution in the last iteration of Algorithms 6–7. FGSA-GA gives better results than the GSA proposed by [22] for all of the functions of Table 7. The maximum improvement is 19 orders of magnitude for the function $F12$. Moreover, FGSA-GA is better than the BGSA proposed in [35] for all of the functions shown in Table 7. The best improvement is of 22 orders of magnitude for the function $F1$. FGSA-GA improves the GSA designed in [27] for the functions $F2$, $F3$, $F4$, $F11$ and $F12$. In particular, for the function $F12$, an improvement of 20 orders of magnitude is achieved. The comparison between our FGSA and FGSA proposed by Khabisi [26] can be accomplished only for the functions $F1 - F4$. Except for the function $F3$, our algorithm has better solutions than FGSA in [26]. FGSA-PSO is better than [22] for the functions $F1$, $F2$, $F3$, $F10$ and $F12$. The best improvement is of 14 orders of magnitude for the function $F1$. Note that, by comparing the FGSA-PSO results with [35], FGSA-PSO provides better outcomes except for $F3$ and $F11$. Only for the function $F12$, FGSA-PSO overcomes the GSA proposed in [27]. FGSA-DE improves the results of [22,26,27,35] for all of the functions in Table 7, except for $F3$ of [26].

Table 8 shows the results of FGSA-GA, FGSA-PSO and FGSA-DE versus GSA of [28] over 16 runs. FGSA-GA improves the results of [28] for all of the functions, providing a relevant improvement for $F1$ of 20 orders of magnitude. The provided solutions by FGSA-PSO are better than GSA [28] only for the function $F1$. FGSA-DE is better than GSA [28] for the functions $F1 - F4$ and $F10$. In this case, an improvement of 18 orders of magnitude for $F1$ is achieved.

Finally, the results show that, both over 30 and 16 runs, FGSA-DE is optimal for unimodal functions, whereas FGSA-GA is good for multimodal functions.

Table 7. Comparison between FGSA-GA and revised GSA over 30 runs.

	GSA [22]	BGSA[35]	GSA[27]	FGSA[26]	FGSA-GA	FGSA-PSO	FGSA-DE
F_1	7.3×10^{-11}	4.65×10^{-5}	8.85×10^{-34}	2.2×10^{-21}	1.78×10^{-27}	7.93×10^{-25}	2.85×10^{-35}
F_2	4.03×10^{-5}	0.0016	1.16×10^{-10}	2.5×10^{-11}	2.32×10^{-13}	6.16×10^{-7}	4.44×10^{-17}
F_3	0.16×10^{3}	26.29	468.44	1.1×10^{-11}	3.27×10^{-5}	5.55×10^{2}	9.68×10^{-4}
F_4	3.7×10^{-6}	1.28	0.0912	4.8×10^{-11}	2.28×10^{-11}	2.85×10^{0}	3.86×10^{-11}
F_{10}	6.9×10^{-6}	0.0400	6.34×10^{-15}	–	7.88×10^{-15}	7.05×10^{-11}	7.99×10^{-15}
F_{11}	0.29	0.00409	4.9343	–	1.30×10^{-9}	1.35×10^{1}	2.89×10^{-2}
F_{12}	0.01	0.9001	0.1103	–	2.44×10^{-21}	1.12×10^{-2}	1.97×10^{-1}

Table 8. Comparison between FGSA-GA and revised GSA over 16 runs.

	GSA [28]	FGSA-GA	FGSA-PSO	FGSA-DE
F_1	8.15×10^{-17}	1.73×10^{-27}	8.91×10^{-25}	2.58×10^{-35}
F_2	1.42×10^{-12}	2.01×10^{-13}	7.26×10^{-7}	5.17×10^{-17}
F_3	0.946	3.22×10^{-5}	5.77×10^{2}	2.58×10^{-4}
F_4	5.62×10^{-5}	8.29×10^{-11}	2.26×10^{0}	3.11×10^{-11}
F_{10}	8.22×10^{-14}	8.44×10^{-15}	6.44×10^{-11}	7.99×10^{-15}
F_{11}	0.043678	1.70×10^{-9}	1.50×10^{1}	6.18×10^{-2}
F_{12}	6.03×10^{-11}	2.04×10^{-21}	1.07×10^{-2}	2.68×10^{-1}

5. Conclusions

The challenge of improving the performances of the search algorithms is an open problem. However, there is no specific algorithm to achieve the best solution of the optimization problems. GSA is a recent search method to find the global optimum. Many researches proposed revised versions of GSA to reduce the time for finding the global optimum. Following this, intelligent techniques have been applied to increase the search performances of GSA. In this paper, some revised FGSA based on the optimization through GA, PSO and DE are proposed. The main idea is to tune the parameter α of the gravitational constant G through a fuzzy controller optimized via GA, PSO and DE. The first FGSA uses GA to optimize the membership functions' slope of the fuzzy variable α. In the second and third FGSA, PSO and DE define the best range of the fuzzy input/output membership functions. In both cases, the optimization process of the fuzzy controller is accomplished just one time: once the fuzzy controller is optimized, it is inserted in the steps of GSA to design FGSA.

In order to compare our outcomes to those of [22,26–28,35], the results are averaged over 30 and 16 runs. The results show that FGSA optimized by GA achieves better results than [22,26–28,35]. The best improvement with respect to [22] is of 19 orders of magnitude. Better results have been achieved by comparing our findings with the findings of [27,35]: a maximum of 22 and 20 orders of magnitude, respectively, is achieved. The order of magnitude of FGSA-GA is six-times greater than [26] for the test function $F1$. The results of [28] are improved for all of the functions achieving a maximum order of magnitude of 10. Moreover, FGSA optimized with PSO achieves better results than GSA-PSO. Significant improvements are achieved for test functions $F1$ and $F7$, where FGSA-PSO is better than GSA-PSO. For the function $F13$, FGSA-DE is better than GSA-DE by two orders of magnitude. Comparing the results of FGSA-GA, FGSA-PSO and FGSA-DE, it follows that FGSA-DE is optimal for unimodal functions, whereas FGSA-GA is good for multimodal functions.

Future studies will focus on novel adaptive gravitational search algorithms for fuzzy-controlled servo system.

Author Contributions: Danilo Pelusi conceived and designed the experiments; Danilo Pelusi performed the experiments; Danilo Pelusi and Raffaele Mascella analyzed the data; Danilo Pelusi, Raffaele Mascella and Luca Tallini contributed analysis tools; Danilo Pelusi wrote the paper. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Eiben, A.E.; Hinterding, R.; Michalewicz, Z. Parameter Control in Evolutionary Algorithms. *IEEE Trans. Evolut. Comput.* **1999**, *3*, 47–75.
2. Montiel, O.; Castillo, O.; Melin, P. ; Rodriguez Diaz, A.; Sepulveda, R. Human evolutionary model: A new approach to optimization. *Inf. Sci.* **2007**, *177*, 2075–2098.
3. Castillo, O.; Valdez, F.; Melin, P. Hierarchical genetic algorithms for topology optimization in fuzzy control systems. *Int. J. Gen. Syst.* **2007**, *36*, 575–591.
4. Holland, J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*; Bradford Books: Cambridge, MA, USA, 1992.
5. Tang, K.S.; Man, K.F.; Kwong, S.; He, Q. Genetic algorithms and their applications. *IEEE Signal Process. Mag.* **1996**, *13*, 22–37.
6. Pelusi, D. Optimization of a fuzzy logic controller using genetic algorithms. In Proceedings of the International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, China, 26–27 August 2011; Volume 2, pp. 143–146.
7. Pelusi, D.; Mascella, R. Optimal control algorithms for second order systems. *J. Comput. Sci.* **2013**, *9*, 183–197.
8. Pelusi, D. On designing optimal control systems through genetic and neuro-fuzzy techniques. In Proceedings of the International Symposium on Signal Processing and Information Technology, Bilbao, Spain, 14–17 December 2011; pp. 134–139.
9. Pelusi, D.; Vazquez, L.; Diaz, D.; Mascella, R. Fuzzy algorithm control effectiveness on drum boiler simulated dynamics. In Proceedings of the 36th International Conference on Telecommunications and Signal Processing, Rome, Italy, 2–4 July 2013; pp. 272–276.
10. Kennedy, J.; Eberhart, R.C. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
11. Ren, Z.; Zhang, A.; Wen, C.; Feng, Z. A scatter learning particle swarm optimization algorithm for multimodal problems, Cybernetics. *IEEE Trans. Cybern.* **2014**, *44*, 1127–1140.
12. Shin, Y.-B.; Kita, E. Search performance improvement of Particle Swarm Optimization by second best particle information. *Appl. Math. Comput.* **2014**, *246*, 346–354.
13. Wu, G.; Qiu, D.; Yu, Y.; Pedrycz, W.; Ma, M.; Li, H. Superior solution guided particle swarm optimization combined with local search techniques. *Expert Syst. Appl.* **2014**, *41*, 7536–7548.
14. Wu, T.H.; Chang, C.C.; Chung, C.H. A simulated annealing algorithm for manufacturing cell formation problems. *Expert Syst. Appl.* **2008**, *34*, 1609–1617.
15. Yu, X.; Zhang, X. Enhanced comprehensive learning particle swarm optimization. *Appl. Math. Comput.* **2014**, *242*, 265–276.
16. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359.
17. Fan, Q.; Yan, X. Self-adaptive differential evolution algorithm with discrete mutation control parameters. *Expert Syst. Appl.* **2015**, *42*, 1551–1572.
18. Fan, Y.; Liang, Q.; Liu, C.; Yan, X. Self-adapting control parameters with multi-parent crossover in differential evolution algorithm. *Int. J. Comput. Sci. Math.* **2015**, *6*, 40–48.
19. Goudos, S.K.; Zaharis, Z.D.; Yioultsis, T.V. Application of a differential evolution algorithm with strategy adaptation to the design of multi-band microwave filters for wireless communications. *Prog. Electromagn. Res.* **2010**, *109*, 123–137.
20. Li, Y.-L.; Zhan, Z.-H.; Gong, Y.-J.; Chen, W.-N.; Zhang, J.; Li, Y. Differential Evolution with an Evolution Path: A DEEP Evolutionary Algorithm. *IEEE Trans. Cybern.* **2015**, *45*, 1798–1810.
21. Li, X.; Yin, M. Application of Differential Evolution Algorithm on Self-Potential Data. *PLoS ONE* **2012**, *7*, e51199.
22. Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. GSA: A Gravitational Search Algorithm. *Inf. Sci.* **2009**, *179*, 2232–2248.
23. Holliday, D.; Resnick, R.; Walker, J. *Fundamentals of Physics*; John Wiley and Sons: Hoboken, NJ, USA, 1993.

24. Askari, H.; Zahiri, S.H. Data Classification Using Fuzzy-GSA. In Proceedings of the International eConference on Computer and Knowledge Engineering, Mashhad, Iran, 13–14 October 2011; pp. 6–11.
25. Zahiri, S.H. Fuzzy gravitational search algorithm an approach for data mining. *Iran. J. Fuzzy Syst.* **2012**, *9*, 21–37.
26. Saeidi-Khabisi, F.S.; Rashedi, E. Fuzzy gravitational search algorithm. In Proceedings of the International E-Conference on Computer and Knowledge Engineering, Mashhad, Iran, 18–19 October 2012; pp.156–160.
27. Sombra, A.; Valdez, F.; Melin, P.; Castillo, O. A new gravitational search algorithm using fuzzy logic to parameter adaptation. In Proceedings of the IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 1068–1074.
28. Amoozegar, M.; Rashedi, E. Parameter Tuning of GSA Using DOE. In Proceedings of the International eConference on Computer and Knowledge Engineering, Mashhad, Iran, 29–30 October 2014.
29. Avishek, P.; Maiti, J. Development of a hybrid methodology for dimensionality reduction in Mahalanobis-Taguchi system using Mahalanobis distance and binary particle swarm optimization. *Expert Syst. Appl.* **2010**, *37*, 1286–1293.
30. Beretaa, M.; Burczynski, T. Comparing binary and real-valued coding in hybrid immune algorithm for feature selection and classification of ECG signals. *Eng. Appl. Artif. Intell.* **2007**, *20*, 571–585.
31. Chuang, L.H.; Chang, H.W..; Tu, C.J. Improved binary PSO for feature selection using gene expression data. *Comput. Biol. Chem.* **2008**, *32*, 29–38.
32. Srinivasa, K.G.; Venugopal, K.R.; Patnaik, L.M. A self-adaptive migration model genetic algorithm for data mining applications. *Inf. Sci.* **2007**, *177*, 4295–4313.
33. Yuan, X.; Nie, H.; Su, A.; Wang, L.; Yuan, Y. An improved binary particle swarm optimization for unit commitment problem. *Expert Syst. Appl.* **2009**, *36*, 8049–8055.
34. Wang, X.; Yang, J.; Teng, X.; Xia, W.; Jensen, R. Feature selection based on rough sets and particle swarm optimization. *Pattern Recognit. Lett.* **2007**, *28*, 459–471.
35. Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. BGSA: Binary Gravitational Search Algorithm. *Nat. Comput.* **2010**, *9*, 727–745.
36. Schutz, B. *Gravity from the Ground Up*; Cambridge University Press: Cambridge, UK, 2003.
37. Shi, Y.; Eberhart, R.; Chen, Y. Implementation of Evolutionary Fuzzy Systems. *IEEE Trans. Fuzzy Syst.* **1999**, *7*, 109–119.
38. Setnes, M.; Roubos, H. GA-Fuzzy Modeling and Classification: Complexity and Performance. *IEEE Trans. Fuzzy Syst.* **2000**, *8*, 509–522.
39. Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed.; Springer-Verlag: New York, NY, USA, 1994.
40. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning. Reading*; Addison-Wesley: Boston, MA, USA, 1989.
41. Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*; MIT Press: Cambridge, MA, USA, 1992
42. Petridis, V.; Kazarlis, S.; Bakirtzis, A. Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems. *IEEE Trans. Syst. Man Cybern.* **1998**, 28, 629.
43. Banzhaf, W.; Nordin, P.; Keller, R.E.; Francone, F.D. *Genetic Programming: An Introduction*; Morgan Kaufmann: San Mateo, CA, USA, 1998.
44. Mitchell, M. *An Introduction to Genetic Algorithms*; The MIT Press: Cambridge, MA, USA, 1999
45. Zitzler, E.; Deb, K.; Thiele, L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.* **2000**, *8* , 173–195.
46. Rudolph, G. *Evolutionary Search under Partially Ordered Sets*; Deptartment Computer Science/LS11, University Dortmund: Dortmund, Germany, 1999.
47. Angeline, P.J. *Two Self-Adaptive Crossover Operators for Genetic Programming, in Advances in Genetic Programming 2*; Angeline, P.J., Kinnear, K.E., Eds.; MIT Press; Cambridge, MA, USA, 1996.
48. Yao, X.; Liu, Y.; Lin, G. Evolutionary programming made faster. *IEEE Trans. Evolut. Comput.* **1999**, *3*, 82–102.