Applied Soft Computing 33 (2015) 292–303

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

The runner-root algorithm: A metaheuristic for solving unimodal and multimodal optimization problems inspired by runners and roots of plants in nature

F. Merrikh-Bayat*

Department of Electrical and Computer Engineering, University of Zanjan, Zanjan, Iran

ARTICLE INFO

Article history:
Received 28 July 2014
Received in revised form 3 April 2015
Accepted 25 April 2015
Available online 5 May 2015

Keywords:
Meta-heuristic optimization algorithm
Nature inspired
Nonparametric statistical analysis
Robust control
Root
Runner

ABSTRACT

This paper proposes a new metaheuristic, the runner-root algorithm (RRA), inspired by the function of runners and roots of some plants in nature. The plants which are propagated through runners look for water resources and minerals by developing runners and roots (as well as root hairs). The first tool helps the plant for search around with random big steps while the second one is appropriate for search around with small steps. Moreover, the plant which is placed at a very good location by chance spreads in a larger area through its longer runners and roots. Similarly, the proposed algorithm is equipped with two tools for exploration: random jumps with big steps, which model the function of runners in nature, and a re-initialization strategy in case of trapping in local optima, which redistributes the computational agents randomly in the domain of problem and models the propagation of plant in a larger area in case of being located in a good position. Exploitation in RRA is performed by the so-called roots and root hairs which respectively apply random large and small changes to the variables of the best computational agent separately (in case of stagnation). Performance of the proposed algorithm is examined by applying it to the standard CEC' 2005 benchmark problems and then comparing the results with 9 state-of-the-art algorithms using nonparametric methods.

1. Introduction

During the past four decades nature had been the source of inspiration for developing new optimization algorithms for solving complicated engineering problems. The first attempts to find a cybernetic solution to a practical problem can be found in the works of Rechenberg [1]. However, the first general-purpose and well-explained algorithm of this type is probably the genetic algorithm (GA) developed by Holland [2]. At this time, various nature-inspired optimization algorithms are available, among them the GA, particle swarm optimization (PSO) [3,4], ant colony optimization (ACO) [5,6], artificial bee colony (ABC) algorithm [7,8], simulated annealing (SA) [9,10], teaching-learning based optimization (TLBO) [11], bacterial foraging optimization (BFO) [12], artificial immune system (AIS) [13], shuffled frog-leaping algorithm (SFLA) [14], differential evolution (DE) [15] and its variants like SaDE [16], imperialist competitive algorithm (ICA) [17], and some others have attracted more attentions. It is a well-known fact that metaheuristic optimization algorithms are capable of solving many complicated engineering problems whose solutions cannot effectively be obtained by using classical (often, gradient-based) optimization algorithms. Numerous successful applications of these algorithms can be found in the literature (see e.g. [18–20]).

Although the source of inspiration is different for nature-inspired optimization algorithms, they still have many similarities. The main relationships between various nature-inspired optimization algorithms are: application of random variables, ability of dealing with uncertain and non-differentiable cost functions, simultaneous application of more than one computational agent for searching the domain of problem, existing a kind of communication scheme between computational agents (e.g., the crossover operator in GA, the social term in PSO, pheromone trail in ACO, dancing of artificial bees in ABC, etc.), and application of the objective function itself rather than its derivative for performing the search. It can be said that a high percent of metaheuristic algorithms actually model the behavior of a certain living things or a certain physical phenomena by performing an optimization. This kind of modeling through optimization is always iterative and makes use of random variables, but the method is not wholly stochastic and has a kind of memory to remember the good solutions of previous iterations and provide

* Tel.: +98 24 3305 4061.
E-mail address: f.bayat@znu.ac.ir

http://dx.doi.org/10.1016/j.asoc.2015.04.048
1568-4946/

the fittest agents of the colony with a more chance to survive and reproduce.

The aim of this paper is to propose a new metaheuristic for solving both unimodal and multimodal optimization problems. The proposed algorithm is inspired by plants such as strawberry and spider plant which propagate through their runners in nature and simultaneously develop roots and root hairs for local search for water resources and minerals (for this reason this algorithm is called the runner-root algorithm (RRA) in the rest of this paper). Similar to some other metaheuristics, RRA does not apply the same number of function evaluations (FE's) at all iterations. More precisely, in RRA the global search for optimal solution (exploration procedure) is performed at all iterations while the local search (exploitation procedure) is performed only when the global search does not lead to a considerable improvement in the value of cost function. Note that the idea of using the runners of plants for developing a metaheuristic algorithm has already been used in the so-called plant propagation algorithm (PPA) [21]. However, the only similarity between the proposed algorithm and PPA is in using the idea of runners.

In order to justify the usefulness of developing a new metaheuristic algorithm we should note that the performance of any algorithm of this type frequently depends on the problem itself, and consequently, a metaheuristic algorithm may be very successful in dealing with a certain problem while it is not so successful in dealing with another one. Hence, the performance of different metaheuristics is often examined by subjecting them to some standard benchmark problems of different type (such as CEC benchmarks [22]) and then analyzing the results by using nonparametric statistical methods [23,24]. Another important fact is that according to NFL theorem [25] there is no *best algorithm*. Hence, any new metaheuristic deserves more study if in dealing with a considerably high percent of benchmark problems it leads to the best results compared to other state-of-the-art algorithms.

The rest of this paper is organized as the following. In Section 2 we briefly review the method of propagation of plants like spider and strawberry plant which can move through their runners. The proposed algorithm is explained in Section 3. Section 4 presents the nonparametric comparisons results of RRA with 9 other state-of-the-art algorithms. A practical open problem in the field of robust control theory is solved using RRA and the results are presented in Section 5. Finally, Section 6 concludes the paper.

(a)

(b)

Fig. 1. (a) Spider plant, (b) Chloris gayana. Fig. 1(b) shows the function of runners and roots as tools for global and local searches, respectively.

2. Source of inspiration for the proposed algorithm: plants with runners and roots in nature

Animals can tolerate environmental changes much better than plants since they have muscle and brain and can decide to move to places with better conditions whenever it is needed. Probably, the most famous examples of this type are the birds which immigrate to warmer places when the weather gets cold. But, the plants are connected to the earth through their roots and cannot move to places with desired conditions anyway. However, some grasses and plants (such as strawberry and spider plant) can be propagated through the so-called *runner* (or *stolon*) as shown in Fig. 1[1] (vegetative propagation). The runner is a creeping stalk produced in the leaf axils and grows out from the *mother* (parent) plant. At the second node of runner a new plant, called *daughter plant*, is formed and then again a new runner arises on the daughter plant to generate another new daughter plant (see Fig. 1). Initially, runner plants produce fewer roots but thereafter put forth excessive fibrous roots and when acquired sufficient growth and roots, the daughter plants can be separated from the mother plant and continue their life individually as the new mother plants. Reproduction of plants such as strawberry can be thought of as a kind of *plant movement* since both the mother and daughter plants have exactly the same genes and they form actually a same plant. However, the mother plant commonly dies sooner than the daughter plant provided that the daughter does not arrive at a location with a very bad condition by chance.

From the mathematical point of view, plants with runners perform a kind of optimization. More precisely, such plants simultaneously perform both the *global* and *local* search to find resources like water and minerals by developing runners and roots (as well as root hairs), respectively. Both the runners and roots are developed almost randomly, but when a runner or a root (hair) arrives at a place with more water resources, the corresponding daughter plant generates more runners, roots, and root hairs, which affects the growth of the whole plant as well. It concludes that if a daughter plant traps in a local optimum point it generates more runners and roots which help it reaching farther places and escaping from that point (this observation is modeled by the *re-initialization* procedure in RRA, which randomly distributes the agents in the domain of problem).

[1] The author of Fig. 1(b) is Macleay Grass Man.

Obviously, in order to arrive at a numerical optimization algorithm inspired by plants like strawberry we need to model the behavior of these plants by simple yet effective rules. In this paper it is assumed that the behavior of such plants can effectively be modeled through the following three facts:

- Each strawberry mother plant is propagated through its runners. These runners rise randomly and each one leads to a new daughter plant (global search with big steps for resources).
- Each strawberry mother plant develops roots and root hairs randomly (local search with relatively big and small steps for resources).
- Strawberry daughter plants which have access to richer resources grow faster, generate more runners and roots, and consequently cover larger areas. On the other hand, the daughter plants which move toward poor resources are more probable to die.

The above rules serve as the source of inspiration for developing the proposed RRA which is discussed with more details in the next section.

3. Mathematical explanation of the runner-root algorithm

Similar to other metaheuristic optimization algorithms, the proposed algorithm begins with a uniformly distributed initial random population in the domain of problem, each of them called a *mother plant*. The number of mother plants is considered equal to N_{pop}. Next, at each iteration any mother plant, except the fittest one, generates a daughter plant (using runners) randomly in the domain of problem (*global search* for better solutions). The distance of each of these daughter plants from the corresponding mother plant is controlled through a constant parameter called d_{runner}(larger d_{runner} larger the distance of daughter plant from its mother plant). The fittest mother plant generates a daughter plant exactly at the same location as itself. Using this procedure N_{pop} daughter plants are generated. In RRA it is assumed that at each iteration each mother plant moves to the location referred to by its daughter plant. If at least one of these daughter plants leads to a *considerable* improvement in the value of cost function compared to the one obtained from the beginning of algorithm, all of the required mother plants of the next iteration are selected among these daughter plants using a combination of elite and roulette-wheel selection method. More precisely, in this case one of the required N_{pop} mother plants of the next iteration is considered equal to the fittest daughter plant of current iteration (elite selection) and the others are obtained by applying the roulette-wheel method to these daughter plants. This statement is equivalent to the fact that while a considerable improvement in the value of cost function is observed, the algorithm continuously investigates the domain of problem for better solutions only through runners (random large steps) which are developed in the vicinity of current solutions (mother plants of current iteration). The amount of this improvement is measured through a parameter called *tol* which is equal to the relative (or absolute) change in the minimum values obtained for cost function in two successive iterations by all daughter plants.

But if none of the resulted daughter plants leads to a considerably better value for cost function, the algorithm begins the *local search* procedure by means of its roots and root hairs. In this case the algorithm assumes that the fittest daughter plant (among those obtained in the current iteration) is located exactly at the same valley as the unknown global best solution, and consequently, applying a random change to each variable of that daughter plant separately and accepting it in case of observing any improvement can move it toward the global best solution. More precisely, if the applied random change to the certain variable of fittest daughter plant leads to a better solution, the algorithm accepts it and applies a random change to the other variable of the resulted modified daughter plant. Else, the previous value of that variable is kept and a random change is applied to the next variable of that daughter plant. This routine is applied to all variables of the fittest daughter plant one by one (note that during the global search procedure, random changes are applied simultaneously to all variables of the selected mother plants while during local search only the variables of the fittest daughter plant are subjected to random changes one by one). Clearly, this strategy guarantees the general movement of the fittest daughter plant toward the closest minimum point located at the same valley, which may be the global best solution. Since in this case it may happen that the fittest daughter plant is located either in a very flat or a narrow valley, both the large and small random changes should be applied to each variable of the best daughter plant. The large and small random changes, which play the role of roots and root hairs in nature, are applied separately and controlled by the constant parameters d_{runner} and d_{root}, respectively (see the following for more details).

After performing the global and local (if necessary) search procedures as mentioned above, mother plants of the next iteration are selected among the resulted daughter plants by using elite and roulette wheel method as described before (note that according to the above discussion, at each iteration only one of the daughter plants, which is the fittest one before applying local search, may be subjected to fine tuning through the local search procedure and the others remain unchanged). To sum up, at each iteration first a global search procedure is performed and one daughter plant is generated for every mother plant. If none of the resulted daughter plants leads to a considerable improvement in the value of cost function, variables of the fittest resulted daughter plant are subjected to random changes one by one by both small and large steps.

The fact that the local search is not applied at all iterations and not to all of the daughter plants can significantly save the number of FE's. However, since it may happen that eventually the algorithm traps in a local optimum point, in order to increase its functionality the algorithm needs to be equipped with a *restart (re-initialization) strategy*. For this purpose, after global and possible local searches, if the relative improvement in the value of cost function in two successive iterations be less than *tol* (as defined before) the value of a counter called *stall_coune* is increased by one, else it is set equal to zero. If the value of this counter reaches the predefined value *stall_max*, the algorithm restarts with a new initial random population. More precisely, in this case all of the solutions obtained so far are discarded and only the best solution obtained in the last iteration is memorized only to be compared with those obtained previously or will be obtained later to determine the final best solution (hence, when the algorithm restarts, no solution is transferred from the last iteration which led to re-initialization to the next one). Note that every time the RRA is used to solve a certain optimization problem it may happen that either it restarts several times or it does not restart at all. For comparing purposes, the fitness evaluations consumed before any re-initialization should be counted toward the cumulative sum. In the following we discuss on different parts of this algorithm with more details.

Consider the following unconstrained optimization problem

$$\min f(\mathbf{x}), \quad \mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u, \tag{1}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is the m–variable cost function to be minimized, $\mathbf{x}^* = arg\min f(\mathbf{x}) \in \mathbb{R}^m$ is the global best solution vector to be calculated, and $\mathbf{x}_l, \mathbf{x}_u \in \mathbb{R}^m$ are two vectors indicating the lower and upper bounds of variables. In case of dealing with a constrained optimization problem one can use standard methods to convert that problem to an equivalent unconstrained one [26]. As mentioned earlier, RRA begins with a set of N_{pop} m-dimensional randomly-generated vectors in the domain of problem each serves

as an initial mother plant. Then, at each iteration any mother plant except the best one generates a random daughter plant and the best one generates a daughter plant exactly equal to itself. Let $\mathbf{x}^k_{mother}(i)$ denotes the location of the k-th mother plant ($k=1,\ldots,N_{pop}$) at the i-th iteration where $\mathbf{x}^1_{mother}(i)$ is equal to the fittest daughter plant of the previous iteration (chosen according to the elite selection) and $\mathbf{x}^k_{mother}(i)$ ($k=2,\ldots,N_{pop}$) are the mother plants selected among the daughter plants of the last iteration using roulette wheel. Hence, using this notation, location of the k-th daughter plant at i-th iteration, $\mathbf{x}^k_{daughter}(i)$, is calculated as

$$\mathbf{x}^k_{daughter}(i)=\begin{cases}\mathbf{x}^1_{mother}(i) & k=1\\ \mathbf{x}^k_{mother}(i)+\mathbf{d}_{runner}\times\mathbf{r}_k, & k=2,\ldots,N_{pop}\end{cases}, \tag{2}$$

where $\mathbf{r}_k\in\mathbb{R}^m$ is a vector whose entries are independent random numbers with uniform distribution in the range $[-0.5, 0.5]$, and d_{runner} is a scalar representing the maximum distance of daughter plants from the mother plant. Note that according to (2) the best daughter plant of the last iteration is necessarily considered as a mother plant, as well as a daughter plant, in the current iteration. Note also that for calculation of any daughter plant a new random vector $\mathbf{r}_k$ must be generated (the reason for using the subscript k for $\mathbf{r}_k$). The value assigned to d_{runner} must be sufficiently large such that it can provide the mother plants trapped in local minimums with a chance to get out of there and further look for the global best solution (which may be in a point as far as $\mathbf{x}_u-\mathbf{x}_l$ form the current local minimum point). Hence, the runners play the very important role of jumping over local minimums, which effectively help the algorithm to avoid trapping in these points. Clearly, in dealing with an optimization problem with known bounds on variables as given in (1) the value of d_{runner} should be comparable with the biggest entry of $\mathbf{x}_u-\mathbf{x}_l$.

The cost function is evaluated at the locations of daughter plants as calculated in (2). If at least one of these daughter plants leads to a significant improvement in the value of cost function compared to the best daughter plant of the previous iteration, i.e., if the inequality

$$\left|\frac{\min\limits_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i))-\min\limits_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i-1))}{\min\limits_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i-1))}\right|\geq tol, \tag{3}$$

is satisfied, the algorithm does not begin the *local search* procedure since (3) means that the global search is still effective (starting the local search procedure when the global search still finds better solutions is not reasonable since local search often consumes a vast number of FE's while its outcome may be dominated by the global search procedure performed in the next iteration). If the cost function is defined such that the denominator of (3) has the chance of being equal to zero, the absolute difference may be used instead of the term in the left hand side of (3). But if (3) is not satisfied, the local search should be done. For this purpose let us denote the best daughter plant among those calculated in (2) as $\mathbf{x}_{daughter,best}(i)$, and the vector obtained by applying a random change to the k-th entry of $\mathbf{x}_{daughter,best}(i)$ as $\mathbf{x}_{perturbed,k}$, i.e.,

$$\mathbf{x}_{perturbed,k}=diag\{1,1,\ldots,1,1+d_{runner}n_k,1,\ldots,1\}\times\mathbf{x}_{daughter,best}(i), \tag{4}$$

where d_{runner} is the same as before, n_k ($k=1,\ldots,m$) is a random number with normal distribution ($\mu=0$ and $\sigma=1$) and $diag\{1,1,\ldots,1,1+d_{runner}n_k,1,\ldots,1\}$ is a diagonal matrix whose all diagonal elements except the k-th one are equal to unity and the k-th one is equal to $1+d_{runner}n_k$. In order to perform the local search, calculate $\mathbf{x}_{perturbed,k}$ for $k=1$ from (4). If $f(\mathbf{x}_{perturbed,1})<f(\mathbf{x}_{daughter,best}(i))$ then $\mathbf{x}_{daughter,best}(i)\leftarrow\mathbf{x}_{perturbed,1}$ else $\mathbf{x}_{daughter,best}(i)$ remains unchanged. Next, calculate $\mathbf{x}_{perturbed,k}$ for $k=2$ from (4) using the resulted $\mathbf{x}_{daughter,best}(i)$. Again if $f(\mathbf{x}_{perturbed,2})<f(\mathbf{x}_{daughter,best}(i))$ then do $\mathbf{x}_{daughter,best}(i)\leftarrow\mathbf{x}_{perturbed,2}$ else keep $\mathbf{x}_{daughter,best}(i)$ unchanged. Repeat this procedure for all of the m variables of $\mathbf{x}_{daughter,best}(i)$. So far the local search with random large steps (model of the function of roots in nature) is completed. Now, a local search with random small steps (model of the function of root hairs in nature) should be treated. This task is performed similar to the local search with random large steps as described before and the only difference is that the following equation is used instead of (4):

$$\mathbf{x}_{perturbed,k}=diag\{1,1,\ldots,1,1+d_{root}r_k,1,\ldots,1\}\times\mathbf{x}_{daughter,best}(i), \tag{5}$$

where d_{root} is a constant scalar often considered much smaller than d_{runner}, $\mathbf{x}_{daughter,best}(i)$ is the final result of local search with random large steps performed before, and r_k ($k=1,\ldots,m$) is a random number with uniform distribution in the range $[-0.5, 0.5]$ (a random number with normal distribution can be used as well). Note that neither for calculation of (4) nor (5) there is a need to do matrix multiplications and these two equations simply mean that only the k-th entry of $\mathbf{x}_{daughter,best}(i)$ should be subjected to a random change.

After performing local search (if necessary) mother plants of the next iteration are selected among the daughter plants of current iteration using a combination of elite and roulette wheel selection. The elite selection can be done simply as the following

$$\mathbf{x}^1_{mother}(i+1)\leftarrow\mathbf{x}_{daughter,best}(i). \tag{6}$$

Before applying the roulette wheel method for selecting the remaining required mother plants of the next iteration, the fitness of k-th daughter plant ($k=1,\ldots,N_{pop}$) is calculated as the following

$$fit(\mathbf{x}^k_{daughter}(i))=\frac{1}{a+f(\mathbf{x}^k_{daughter}(i))-f(\mathbf{x}_{daughter,best}(i))}, \tag{7}$$

where a is a positive real constant controlling the selection pressure. Note that according to (7) the best daughter plant receives the maximum fitness which is equal to $1/a$ and the other daughter plants receive smaller fitness values. After calculating the fitness values, the probability of choosing the k-th daughter plant of current iteration as a mother plant of next iteration, p_k, is calculated as follows

$$p_k=\frac{fit(\mathbf{x}^k_{daughter}(i))}{\sum\limits_{j=1}^{N_{pop}} fit(\mathbf{x}^j_{daughter}(i))}. \tag{8}$$

Note that many other methods (such as the so-called *linear ranking* method often used in GA) can also be used instead of (7) for calculating the fitness of daughter plants. Note also that in dealing with (7) we have $\mathbf{x}^k_{daughter}(i)=\mathbf{x}_{daughter,best}(i)$ for some k. The algorithm may also get restarted as described before.

Pseudo-code of RRA is presented in Table 1. In this table *rand* represents a random number with uniform distribution within the range $[-0.5, 0.5]$, each entry of $\mathbf{r}_k\in\mathbb{R}^m$ is defined the same as *rand*, and n_k ($k=1,\ldots,m$) is a random number with normal distribution ($\mu=0$ and $\sigma=1$).

3.1. Discussion on the exploration and exploitation ability of RRA

Roughly speaking, *exploration* and *exploitation* refer to the ability of an algorithm to visit entirely new regions of search space and searching around the previous solutions, respectively (see [27] for a survey on this subject). It is a common belief among experienced evolutionary algorithms researchers that the functionality

Table 1
Pseudo-code of the runner-root algorithm (RRA).

Initialize d_{runner}, d_{root}, N_{pop}, *stall_max*, *tol*, *a*
$\mathbf{x}^k_{mother}(1) \leftarrow \mathbf{x}_l + rand \times (\mathbf{x}_u - \mathbf{x}_l)$ for $k = 1, \ldots, N_{pop}$ //initial random mother plants
$stall\_count \leftarrow 0$, $i \leftarrow 1$
REPEAT until termination conditions are not met
$\mathbf{x}^k_{daughter}(i) = \begin{cases} \mathbf{x}^1_{mother}(i) & k = 1 \\ \mathbf{x}^k_{mother}(i) + d_{runner} \times r_1, & k = 2, \ldots, N_{pop} \end{cases}$ for $k = 1, \ldots, N_{pop}$
$\mathbf{x}_{daughter,best}(i) \leftarrow \underset{x=\mathbf{x}^k_{daughter}(i)}{arg\,\min}\ f(\mathbf{x})$ //consumes N_{pop} function evaluations
IF $i > 1$ AND $\left| \frac{\min_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i)) - \min_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i-1))}{\min_{k=1,\ldots,N_{pop}} f(\mathbf{x}^k_{daughter}(i-1))} \right| < tol$ **THEN**
FOR k FROM 1 UNTIL N_{pop} **DO**//local search with large steps
$\mathbf{x}_{perturbed,k} \leftarrow diag\{1, 1, \ldots, 1, 1 + d_{runner} n_k, 1, \ldots, 1\} \times \mathbf{x}_{daughter,best}(i)$
IF $f(\mathbf{x}_{perturbed,k}) < f(\mathbf{x}_{daughter,best}(i))$ **THEN**//consumes a function evaluation
$\mathbf{x}_{daughter,best}(i) \leftarrow \mathbf{x}_{perturbed,k}$
END
END (k-loop)
FOR k FROM 1 UNTIL N_{pop} **DO**//local search with small steps
$\mathbf{x}_{perturbed,k} \leftarrow diag\{1, 1, \ldots, 1, 1 + d_{root} r_k, 1, \ldots, 1\} \times \mathbf{x}_{daughter,best}(i)$
IF $f(\mathbf{x}_{perturbed,k}) < f(\mathbf{x}_{daughter,best}(i))$ **THEN**//consumes a function evaluation
$\mathbf{x}_{daughter,best}(i) \leftarrow \mathbf{x}_{perturbed,k}$
END
END (k-loop)
END (if)
$\mathbf{x}^1_{mother}(i+1) \leftarrow \mathbf{x}_{daughter,best}(i)$
Calculate the fitness of k-th daughter plant from $fit(\mathbf{x}^k_{daughter}(i)) \leftarrow \frac{1}{a + f(\mathbf{x}^k_{daughter}(i)) - f(\mathbf{x}_{daughter,best}(i))}$ and the probability of choosing it from $p_k = \frac{fit(\mathbf{x}^k_{daughter}(i))}{\sum_{j=1}^{N_{pop}} fit(\mathbf{x}^j_{daughter}(i))}$ for $k = 1, \ldots, N_{pop}$
FOR k FROM 2 UNTIL N_{pop} **DO**//generating the mother plants of next iteration
$\mathbf{x}^k_{mother}(i+1) \leftarrow \mathbf{x}^{ind}_{daughter}(i)$ where *ind* is the index of the daughter plant selected among the daughter plants of current iteration using roulette wheel
END (k-loop)
IF $\left| \frac{f(\mathbf{x}_{daughter,best}(i)) - f(\mathbf{x}_{daughter,best}(i-1))}{f(\mathbf{x}_{daughter,best}(i-1))} \right| < tol$ **THEN**//checking for stall condition
$stall\_count \leftarrow stall\_count + 1$
ELSE
$stall\_count \leftarrow 0$
END
IF $stall\_count > stall\_max$ **THEN** restart the algorithm (that is, memorize the best solution obtained in the current iteration for comparing purposes, go to the second line of algorithm and discard all the solutions obtained in the current iteration. The function evaluations consumed before re-initialization counted toward cumulative.)
$i \leftarrow i + 1$
END (repeat)

of any algorithm of this type strictly depends on its exploration and exploitation abilities and the method used to establish a good ratio between them. It is also believed that the differences between various metaheuristic optimization algorithms, as well as the functionality of a certain algorithm, can be explained by the exploration and exploitation methods and the rate of applying them. In the following we briefly study the exploration and exploitation ability of RRA.

According to the previous discussion RRA is equipped with two tools for exploitation: first, the roots and root-hairs, which are specially designed for more search around the best solution of current iteration, and second, the elite selection schema which guarantees the transmission of the best solution of current iteration to the next iteration, and consequently, doing further searches around it. The proposed RRA also makes use of two different strategies for exploration. First, at each iteration all variables of each mother plant (except the best one) are simultaneously subjected to random changes with large steps. The length of these steps should be considered sufficiently large such that the computational agents have a chance to move to any point in the search space. Second, if no considerable improvement is observed in the minimum value obtained for cost function after certain number of successive iterations, the algorithm restarts with an initial random population uniformly distributed in the domain of problem (re-initialization). This re-initialization strategy is very important and effective in the proposed algorithm since it is observed that in dealing with some of the complicated multimodal benchmark problems this algorithm (without providing it with the re-initialization capability) has a good chance to find the global best solution, but if it cannot find it consuming more FE's does not help it anymore. In this case re-initialization can promote the exploration and highly increases the chance of finding the global best solution.

In RRA the ratio of exploration and exploitation is managed by control parameters of algorithm. Although any parameter may implicitly affect the exploration-exploitation ability of algorithm, according to the sensitivity analysis presented in Section 4.3 and the simulation results presented in Section 4 the parameters *stall_max* and *tol* have the most significant effect on the balance between exploration and exploitation. More precisely, in general it is expected that increasing *stall_max* or decreasing *tol* leads to more exploitations since it decreases the chance of re-initialization. Moreover, according to the discussions in [27] RRA applies an adaptive schema for controlling the balance between exploration and exploitation since re-initialization happens when the best fitness does not considerably change for certain number of successive iterations.

4. Nonparametric statistical analysis results

In this section we apply nonparametric (or distribution-free) inferential statistical methods to compare the performance of RRA with some other state-of-the-art metaheuristic algorithms. Nonparametric methods are mathematical techniques for statistical hypothesis testing which, unlike parametric statistics, make no assumptions (such as normality or homoscedasticity) about the probability distributions of the variables being assessed. These methods are especially useful when the outcomes of stochastic experiments are ordinal, ranked, subject to outliers or measured imprecisely. Currently, various nonparametric methods are available which can be used both for pairwise and multiple comparisons. Among others, the Sign test and Wilcoxon test (for pairwise comparisons) and Multiple sign test, Friedman test, Friedman aligned ranks test and Quade test (for multiple comparisons) are more widely used in the literature. The seminal paper [24] is especially devoted to the application of nonparametric statistical tests for comparing the performance of evolutionary and swarm intelligence algorithms. The data used in the following discussion is exactly adopted from [24], and the readers are referred to that paper for more information on this subject and also for the values assigned to the control parameters of the algorithms discussed in the following.

Before applying any nonparametric stochastic test for comparing the performances of algorithms under consideration, one needs to solve a considerably large number of benchmark problems by them. As a general rule, in multiple comparisons the number of algorithms used must be lower than the number of benchmark problems [24], which is fulfilled in the current study. More precisely, here the comparison is made between 10 different continuous optimization algorithms, i.e., PSO, IPOP-CMA-ES [28], CHC [29], SSGA [30,31], SS-Arit [32], SS-BLX [33], DE-Exp, DE-Bin [15], SaDE [16], and RRA. For this purpose these algorithms are used to solve 25 different test problems of dimension 10 appeared in the CEC'2005 special section on real parameter optimization [22]. This set of benchmark problems consists of a wide variety of

Table 2
Average error obtained in the 25 benchmark functions (exact replication of [24]).

Func.	PSO	IPOP-CMA-ES	CHC	SSGA	SS-BLX	SS-Arit	DE-Bin	DE-Exp	SaDE
F1	1.234E−04	0.000E+00	2.464E+00	8.420E−09	3.402E+01	1.064E+00	7.716E−09	8.260E−09	8.416E−09
F2	2.595E−02	0.000E+00	1.180E+02	8.719E−05	1.730E+00	5.282E+00	8.342E−09	8.181E−09	8.208E−09
F3	5.174E+04	0.000E+00	2.699E+05	7.948E+04	1.844E+05	2.535E+05	4.233E+01	9.935E+01	6.560E+03
F4	2.488E+00	2.932E+03	9.190E+01	2.585E−03	6.228E+00	5.755E+00	7.686E−09	8.350E−09	8.087E−09
F5	4.095E+02	8.104E−10	2.641E+02	1.343E+02	2.185E+00	1.443E+01	8.608E−09	8.514E−09	8.640E−09
F6	7.310E+02	0.000E+00	1.416E+06	6.171E+00	1.145E+02	4.945E+02	7.956E−09	8.391E−09	1.612E−02
F7	2.678E+01	1.267E+03	1.269E+03	1.271E+03	1.966E+03	1.908E+03	1.266E+03	1.265E+03	1.263E+03
F8	2.043E+01	2.001E+01	2.034E+01	2.037E+01	2.035E+01	2.036E+01	2.033E+01	2.038E+01	2.032E+01
F9	1.438E+01	2.841E+01	5.886E+00	7.286E−09	4.195E+00	5.960E+00	4.546E+00	8.151E−09	8.330E−09
F10	1.404E+01	2.327E+01	7.123E+00	1.712E+01	1.239E+01	2.179E+01	1.228E+01	1.118E+01	1.548E+01
F11	5.590E+00	1.343E+00	1.599E+00	3.255E+00	2.929E+00	2.858E+00	2.434E+00	2.067E+00	6.796E+00
F12	6.362E+02	2.127E+02	7.062E+02	2.794E+02	1.506E+02	2.411E+02	1.061E+02	6.309E+01	5.634E+01
F13	1.503E+00	1.134E+00	8.297E+01	6.713E+01	3.245E+01	5.479E+01	1.573E+00	6.403E+01	7.070E+01
F14	3.304E+00	3.775E+00	2.073E+00	2.264E+00	2.796E+00	2.970E+00	3.073E+00	3.158E+00	3.415E+00
F15	3.398E+02	1.934E+02	2.751E+02	2.920E+02	1.136E+02	1.288E+02	3.722E+02	2.940E+02	8.423E+01
F16	1.333E+02	1.170E+02	9.729E+01	1.053E+02	1.041E+02	1.134E+02	1.117E+02	1.125E+02	1.227E+02
F17	1.497E+02	3.389E+02	1.045E+02	1.185E+02	1.183E+02	1.279E+02	1.421E+02	1.312E+02	1.387E+02
F18	8.512E+02	5.570E+02	8.799E+02	8.063E+02	7.668E+02	6.578E+02	5.097E+02	4.482E+02	5.320E+02
F19	8.497E+02	5.292E+02	8.798E+02	8.899E+02	7.555E+02	7.010E+02	5.012E+02	4.341E+02	5.195E+02
F20	8.509E+02	5.264E+02	8.960E+02	8.893E+02	7.463E+02	6.411E+02	4.928E+02	4.188E+02	4.767E+02
F21	9.138E+02	4.420E+02	8.158E+02	8.522E+02	4.851E+02	5.005E+02	5.240E+02	5.420E+02	5.140E+02
F22	8.071E+02	7.647E+02	7.742E+02	7.519E+02	6.828E+02	6.941E+02	7.715E+02	7.720E+02	7.655E+02
F23	1.028E+03	8.539E+02	1.075E+03	1.004E+03	5.740E+02	5.828E+02	6.337E+02	5.824E+02	6.509E+02
F24	4.120E+02	6.101E+02	2.959E+02	2.360E+02	2.513E+02	2.011E+02	2.060E+02	2.020E+02	2.000E+02
F25	5.099E+02	1.818E+03	1.764E+03	1.747E+03	1.794E+03	1.804E+03	1.744E+03	1.742E+03	1.738E+03

real parameter optimization problems including 5 unimodal functions (F1: shifted Sphere function, F2: shifted Schwefel's problem, F3: shifted rotated high conditioned Elliptic function, F4: shifted Schwefel's problem with noise in fitness, F5: Schwefel's problem with global optimum on bounds), and 20 multimodal functions (F6: shifted Rosenbrock's function, F7: Shifted rotated Griewank function without bounds, F8: shifted rotated Ackley's function with global optimum on bounds, F9: shifted Rastrigin's function, F10: shifted rotated Rastrigin's function, F11: shifted rotated Weierstrass function, F12: Schwefel's problem, F13: expanded extended Griewank's plus Rosenbrock's function (F8F2), F14: shifted rotated expanded Scaffers F6, F15–F25: hybrid functions defined through compositions of 10 out of 14 previous functions). More details on the benchmark problems and the settings of the parameters used in algorithms can be found in [22] and [24], respectively. In [24] all of the above algorithms (except RRA) have been applied 50 times to each test function, where each of these runs stops either after 100,000 FE's or when the error obtained is less than 10^{-8}. Table 2, which is the exact replication of the data presented in [24], shows the average error obtained for each algorithm (except RRA) over the 25 benchmark functions under consideration. Note that the reason why some entries in Table 2 are less than 10^{-8} is that the last improvement was so big (especially when IPOP-CMA-ES is used to solve F1–F3 and F6).

The proposed RRA is also subjected to the similar experiment as described above. For this purpose, in order to show the effect of re-initialization strategy on the performance of RRA, first each of the benchmark problems under consideration is solved 50 times independently (each of these runs possibly consists of several re-initializations) assuming a certain value for the control parameter *stall_max* as well as others. Then each of these problems is solved again 50 times independently assuming another value for *stall_max* (other control parameters are considered exactly the same in two experiments), and finally the averaged results obtained in two cases are compared. For this purpose, first the control parameters of RRA are considered as $d_{root} = 10^{-3}$, $d_{runner} = 3$, $N_{pop} = 50$, $a = 10^{-1}$, $stall\_max = 250$, and $tol = 10^{-3}$, and the next time all of them are considered as before except *stall_max* which is considered equal to 100 (the effect of re-initialization strategy could also be studied by changing the value of *tol* instead of *stall_max*, however, changing *stall_max* seems more effective). The results obtained by applying RRA to CEC'2005 benchmark problems as described above are shown in Table 3. Note that, as it is observed in this table, it is expected that in dealing with unimodal problems increasing the value assigned to *stall_max* (i.e., decreasing the rate of re-initializations) improves the quality of results, which is generally achieved at the cost of decreasing the performance of algorithm in dealing with multimodal problems. All of the comparisons presented in the next sections are treated using the data presented in Table 2 and in the rightmost column of Table 3 (obtained assuming *stall_max* = 100).

Table 3
Average error obtained in the 25 benchmark functions using RRA when applied with two different settings for control parameters (each data is averaged over 50 independent runs).

Func.	RRA (*stall_max* = 250)	RRA (*stall_max* = 100)
F1	7.000E−09	3.198E−07
F2	3.361E−03	1.179E−01
F3	6.164E+04	6.282E+04
F4	1.270E−02	5.263E−02
F5	5.229E−01	1.538E+00
F6	3.001E+01	7.535E+01
F7	4.281E+00	5.416E+01
F8	2.010E+01	2.007E+01
F9	1.301E−07	6.860E−03
F10	2.466E+01	2.228E+01
F11	6.062E+00	6.028E+00
F12	1.326E+02	1.742E+02
F13	3.591E−01	4.515E−01
F14	3.648E+00	3.838E+00
F15	8.280E+01	1.541E+01
F16	1.431E+02	1.390E+02
F17	1.347E+02	1.188E+02
F18	6.022E+02	6.291E+02
F19	6.435E+02	6.192E+02
F20	6.673E+02	6.451E+02
F21	5.942E+02	3.711E+02
F22	7.349E+02	7.905E+02
F23	7.093E+02	5.202E+02
F24	2.000E+02	2.000E+02
F25	3.604E+02	3.520E+02

Table 4
Wilcoxon signed ranks test results.

Comparison	R^+	R^-	p-value
RRA versus CHC	289	36	0.00034
RRA versus SS-Arit	275	50	0.00126
RRA versus SS-BLX	262	63	0.00379
RRA versus PSO	260	65	0.00444
RRA versus SSGA	250	75	0.00945
RRA versus IPOP-CMA-ES	194	131	0.20022
RRA versus DE-Exp	149	176	0.27469
RRA versus SaDE	140	185	0.32635
RRA versus DE-Bin	134	166	0.36073

4.1. Pairwise comparisons results

The pairwise comparisons can be used to compare the performance of two algorithms when applied to a common set of benchmark problems. Two well-known methods for this purpose are the Sign test and the Wilcoxon signed ranks test. The latter is used in the following to detect the possible significant differences between two algorithms since it is believed that Wilcoxon signed ranks test is safer and more robust compared to the sign test. In the following, RRA is considered as the control whose performance is compared with the other 9 algorithms under consideration.

4.1.1. The Wilcoxon signed ranks test

In the following first we briefly explain the Wilcoxon signed ranks test and then apply it to our problem. Let N be the sample size (i.e., the number of pairs which is equal to 25 in our study), which means that there are a total of $2N$ data points. Moreover, let $x_{1,i}$ and $x_{2,i}$ ($i = 1, \ldots, N$) denote the measurements. The Wilcoxon signed ranks test consists of the following five steps:

1. For $i = 1, \ldots, N$, calculate $\left|x_{2,i} - x_{1,i}\right|$ and $\mathrm{sgn}(x_{2,i} - x_{1,i})$, where sgn is the sign function.
2. Exclude pairs with $\left|x_{2,i} - x_{1,i}\right| = 0$. Let N_r denotes the reduced sample size.
3. Order the remaining N_r pairs from smallest absolute difference to the largest absolute difference.
4. Rank the pairs, starting with the smallest as 1. Ties receive a rank equal to the average of the ranks they span. Let R_i denote the rank.
5. Calculate the test statistic W, which is equal to the absolute value of the sum of the signed ranks as the following

$$W = \left| \sum_{i=1}^{N_r} \left[sgn(x_{2,i} - x_{1,i}) \times R_i \right] \right| = \left| R^+ - R^- \right| \tag{9}$$

where R^+ and R^- represent the sum of ranks for the problems in which the second algorithm outperforms the first one and the first algorithm outperforms the second one, respectively. As a well-known fact, as N_r increases, the sampling distribution of W converges to a normal distribution. For $N_r \geq 10$ the distribution of W can safely be considered as normal distribution with the z-score $z = (W - 0.5)/\sigma_w$ where

$$\sigma_w = \sqrt{\frac{N_r(N_r+1)(2N_r+1)}{6}} \tag{10}$$

Table 4 summarizes the results for Wilcoxon signed rank tests. In this table the (unadjusted) p-values are calculated using the procedure described above. However, these p-values are not suitable to extract a conclusion since the same data (the data for RRA) is tested multiple times. More precisely, an accumulated error is resulted from the combination of pairwise comparisons if we draw a conclusion comprising more than one pairwise comparison. Hence, when the same data is tested multiple times, a post hoc test must be applied to control the Family-Wise Error Rate (FWER), which is defined as the probability of making one or more false discoveries among all the hypotheses when performing multiple hypotheses tests [24]. In the next section we apply a post hoc analysis to determine which of the p-values presented in Table 4 are truly statistically significant.

Table 5
Adjusted and unadjusted p-values for Wilcoxon signed ranks test presented in Table 4.

Comparison	Unadjusted p-value	Adjusted p-value (Holm procedure)
RRA versus CHC	0.00034	0.00307
RRA versus SS-Arit	0.00126	0.01010
RRA versus SS-BLX	0.00379	0.02651
RRA versus PSO	0.00444	0.02663
RRA versus SSGA	0.00945	0.04723
RRA versus IPOP-CMA-ES	0.20022	0.80087
RRA versus DE-Exp	0.27469	0.82408
RRA versus SaDE	0.32635	0.82408
RRA versus DE-Bin	0.36073	0.82408

4.1.2. Post hoc analysis using Holm procedure

In this section the well-known Holm method is used for post hoc analysis (see [24] for a review of post hoc analysis methods including Holm's method). To briefly explain this method let p_1, $p_2, \ldots, p_{k-1}$ be the ordered p-values such that $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$ ($k = 10$ for the data provided in Table 4). In the Holm procedure the i-th adjusted p-value (APV_i) is equal to $\min\{v, 1\}$ where $v = \max\{(k-j)p_j : 1 \leq j \leq i\}$, $i = 1, \ldots, k-1$. The resulted APV's can be compared directly with the chosen significance level α. More precisely, Holm procedure starts with the most significant p-value, i.e. APV_1. If APV_1 is below α the corresponding hypothesis is rejected and we are allowed to compare APV_2 with α. If the second hypothesis is rejected, the test proceeds with the third one, and so on. Once a certain null hypothesis cannot be rejected, all of the remaining hypotheses are retained as well [24].

Table 5 shows unadjusted and adjusted p-values (using Holm procedure) for the data given in Table 4. According to the results presented in this table RRA shows an improvement in pairwise comparisons over CHC, SS-Arit, SS-BLX, PSO, and SSGA with the level of significance $\alpha = 0.05$. However, no improvement is observed when RRA is compared with IPOP-CMA-ES, DE-Exp, SaDE, and DE-Bin.

4.2. Multiple comparisons with a control method

Multiple comparisons tests can be used to carry out a comparison which involves more than two algorithms. These tests are divided into two main categories: $1 \times N$ and $N \times N$ comparisons tests. In the following we discuss only on $1 \times N$ comparisons tests, which can be used to highlight a control method among others. The Friedman, Friedman aligned ranks, and Quade tests are three well-known $1 \times N$ comparisons tests used in the literature. The procedure of these tests can be found in [24]. In the following we will only report the results obtained by using the standard Friedman test followed by the Holm post hoc procedure.

Table 6 depicts the average ranks achieved by the Friedman test according to the data provided in Table 2 and the rightmost column of Table 3. Using this test the algorithm which achieves a smaller rank is suspected to be a better performing algorithm (hence DE-Exp is suspected to be the best performing algorithm here). In Table 6 the p-value obtained through the statistics of the test strongly suggests the existence of significant differences among the algorithms considered. However, since the results in

Table 6
Ranks achieved by the Friedman test in the case of study.

Algorithms	Friedman
DE-Exp	3.92
DE-Bin	4.16
SaDE	4.38
RRA	4.86
IPOP-CMA-ES	5.08
SS-BLX	5.56
SSGA	6.20
SS-Arit	6.24
CHC	7.12
PSO	7.48
Statistic	37.4138
p-value	2.2199×10^{-5}

Table 7
Adjusted and unadjusted p-values for the Friedman test (RRA is the control method).

Friedman	Unadjusted p-value	Adjusted p-value (Holm procedure)
PSO	0.001108	0.009971
CHC	0.004154	0.033235
SS-Arit	0.053526	0.374683
SSGA	0.058806	0.374683
SS-BLX	0.206830	1.000000
IPOP-CMA-ES	0.398620	1.000000
SaDE	0.712448	1.000000
DE-Bin	0.793170	1.000000
DE-Exp	0.863842	1.000000

this table are obtained by multiple uses of the same data, a post hoc test must be applied to control FWER and before extracting any conclusion. For this purpose, first the (unadjusted) p-values corresponding to the Friedman ranks presented in Table 6 are calculated (see [24] for details) and then the corresponding adjusted p-values are obtained using the Holm procedure (the same as the one discussed in Section 4.1.2.). The results are presented in Table 7. As we can see in this table the Friedman test shows an improvement of RRA over PSO and CHC with the level of significance $\alpha = 0.05$. However, no improvement of RRA over the remaining 7 algorithms can be concluded.

4.3. Sensitivity analysis on control parameters of RRA

In the following we present the sensitivity analysis on control parameters of RRA. The reason for usefulness of this analysis is that it indicates the amount of robustness of algorithm to changes in control parameter settings. Clearly, smaller sensitivity is more desired since it means easier tuning procedure. For this purpose, F9 and F25 are chosen among the 25 different benchmark problems studied at the beginning of this section and the effect of changing the control parameters on the average error of these problems is studied (this approach is similar to the method used in [34] for sensitivity analysis). The reason for choosing F25 is that according to the data provided in Tables 2 and 3 RRA leads to the best results compared to other algorithms in dealing with this problem. Hence, the following discussion aims to show that this superiority is preserved after changing the values assigned to control parameters of RRA in this simulation. The reason for choosing F9 is that, as it is observed in Tables 2 and 3, in dealing with this problem RRA does not lead to the best results. Hence, the following sensitivity analysis aims to show that this algorithm is capable of performing considerably better by appropriate choice of its control parameters.

Figs. 2 and 3 show the sensitivity analysis results for F9 and F25, respectively. At each figure the value of control parameter is changed around its nominal value and then the corresponding value of average error is calculated (the horizontal line at each figure passes through the nominal value, where the values $d_{root} = \{10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}\}$, $a = \{0.01, 0.05, 0.1, 0.5, 1\}$, and $tol = \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ are used in simulations). Fig. 2 clearly shows that the average error in this example is highly affected by *stall_max* which controls the exploration ability of algorithm through the rate of re-initializations. In this example increasing the value assigned to *stall_max* decreases the rate of re-initializations, and consequently, provides the algorithm with a chance for more exploitation (more searches around the best solution obtained so far), which often improves the accuracy of results in dealing with unimodal and some multimodal problems. It is also observed that, as it is expected, decreasing the value assigned to *tol* has almost the same effect as increasing *stall_max* but with a lower severity.

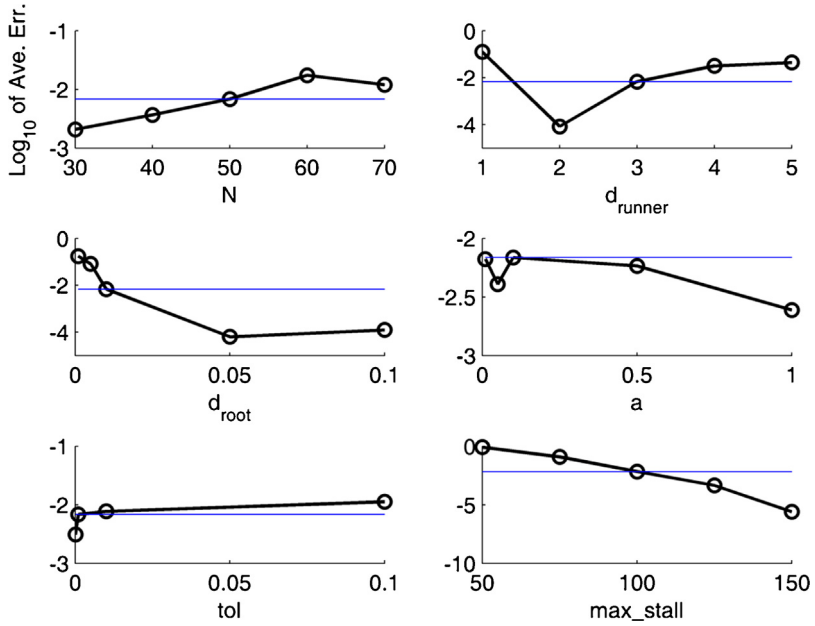

Fig. 2. Plot of base 10 logarithm of average error versus control parameters corresponding to the benchmark problem F9 solved using RRA (horizontal line indicates the operating point).

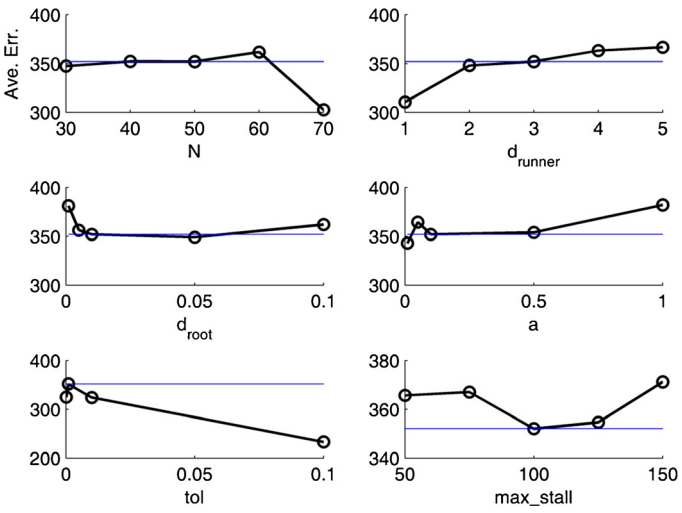

Fig. 3. Plots of average errors versus control parameters corresponding to the benchmark problem F25 solved using RRA (horizontal line indicates the operating point).

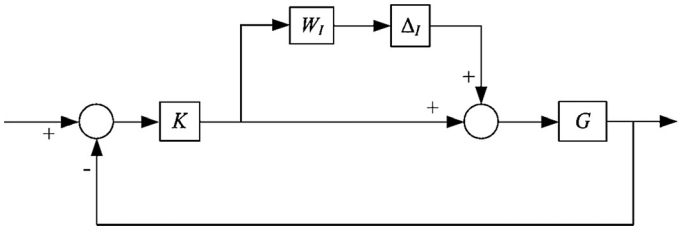

Fig. 4. A feedback system with multiplicative uncertainty at the process input.

Fig. 3 shows the sensitivity analysis results for F25. The main point in relation to this figure is that, despite the variations in control parameters, the algorithm preserves its superiority over other algorithms. In this figure it is observed that increasing the value of *tol*, and consequently, increasing the rate of re-initialization significantly improves the accuracy of results.

5. Application of RRA for solving an open problem in the field of robust control theory

In this example we introduce an unsolved problem in the field of robust control theory and provide a numerical solution for it using RRA. Consider the single-input single-output (SISO) feedback system shown in Fig. 4 where the uncertainty of process is modeled with the full multiplicative input uncertainty ($\left\|\Delta_I\right\|_\infty \le 1$) [35]. In this figure assuming that the process nominal transfer function, $G(s)$, and uncertainty weight, $w_I(s)$, are stable and known, the controller which simultaneously satisfies nominal performance (NP), robust stability (RS), and robust performance (RP) is obtained by solving the following H_∞-norm inequality [35]:

$$\left\||w_P S| + |w_I T|\right\|_\infty < 1, \tag{11}$$

where $S=(1+KG)^{-1}$, $T=KG(1+KG)^{-1}$, K is the unknown controller, and w_p is the stable weight function used to determine the desired performance (i.e., it is assumed that the nominal feedback system with $\Delta_I=0$ has a desired performance if and only if we have $|S| < |w_P|^{-1}, \forall\omega$).

Note that the only unknown parameter in (11) is the transfer function of controller, K. As a classical result [35], the controller which satisfies (11) also guarantees NP, RS, and RP, but the nominal stability (NS) is not guaranteed in this case, i.e., the controller satisfies (11) may result in an unstable feedback system. Hence, any solution for (11) must also be checked from the NS point of view. In the following, we will consider the NS condition as a constraint for a minimization problem.

At this time there is no method available to find a K which simultaneously satisfies (11) and guarantees the stability of nominal feedback system [35]. Instead, algorithms like *hinfsyn* in Matlab, find K such that the following closely related mixed sensitivity H_∞ condition is satisfied

$$\left\|\begin{matrix} w_p S \\ w_I T \end{matrix}\right\|_\infty = \max_\omega \sqrt{|w_p S|^2 + |w_I T|^2} < 1. \tag{12}$$

Although the above condition is within a factor of at most $\sqrt{2}$ to condition (11), they are not exactly the same [35]. Note that similar to (11), the NS is not included in (12). The very important point in relation to (11) (as well as (12)) is that it may or may not have a solution. However, when it has, often it is not unique (infinity many solutions exist in general). For that reason, this problem is often formulated as a minimization problem in which $\gamma \triangleq \left\||w_P S| + |w_I T|\right\|_\infty$ (or $\gamma \triangleq \left\|\begin{matrix} w_p S \\ w_I T \end{matrix}\right\|_\infty$ in dealing with (12)) is minimized by suitable choice of the (stabilizing) K. According to the above discussion it is obvious that the robust control problem under consideration really has a solution if for some stabilizing K we have $\gamma = \left\||w_P S| + |w_I T|\right\|_\infty < 1$.

The aim of this paper is to calculate K directly from (11). For this purpose we find K by solving the following constrained minimization problem:

$$\begin{matrix} \min\limits_K \gamma = \left\||w_P S| + |w_I T|\right\|_\infty \\ s.t. \quad 1 + KG(s) \neq 0,\ Re\{s\} \ge 0 \end{matrix}, \tag{13}$$

where the constraint $1+KG(s) \neq 0$, $\mathrm{Re}\{s\} \ge 0$ represents the stability of the nominal closed-loop system. A classical approach for solving constrained optimization problems is to express the problem as an equivalent one without any constraints (a survey of techniques used for handing the constraints can be found in [26]). As the simplest method, the solution of (13) can be obtained by solving the following unconstrained optimization problem

$$\min F(\mathbf{x}) \triangleq \left\||w_P S| + |w_I T|\right\|_\infty + \lambda \langle g(\mathbf{x}) \rangle, \tag{14}$$

where $\mathbf{x}$ is a vector containing the unknown parameters of controller, $\lambda > 0$ is the penalty factor, $g(\mathbf{x})$ is equal to the real part of the rightmost pole of the nominal closed-loop system (i.e., the system of Fig. 13 when $\Delta_I = 0$), and $\langle . \rangle$ is the bracket function defined as the following

$$\langle g(\mathbf{x}) \rangle \triangleq \begin{cases} g(\mathbf{x}) & g(\mathbf{x}) \ge 0 \\ 0 & g(\mathbf{x}) < 0 \end{cases}. \tag{15}$$

Note that according to (14) and (15), when the candidate solution $\mathbf{x}$ does not violate the stability condition, the real part of the rightmost pole of the nominal closed-loop system is negative, and consequently, the cost function is simply equivalent to $F(\mathbf{x}) = \left\||w_P S| + |w_I T|\right\|_\infty$. But, when the stability condition is violated, the second term in the right hand side of (14) puts an extra force on the algorithm to obtain solutions that lead to stable feedback systems. Clearly, larger the value of λ, higher the force on algorithm to find the solutions that stabilize the feedback system.

In the following we use the above procedure to find a controller for a process with nominal transfer function $G(s)=1000/(0.1s^2+s)$ assuming $w_I = 0.2$ and $w_P = (0.2s+1)/(s+0.001)$. In this example $G(s)$ is the approximate model of a DC motor and $w_I = 0.2$ represents the fact that this model has 20% uncertainty at all frequencies. Considering the fact that the order of K is at least equal to the order

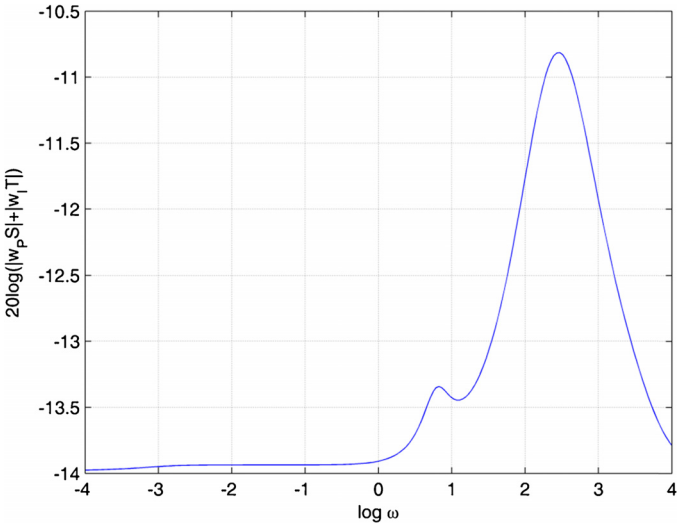

Fig. 5. Plot of $|w_P S| + |w_I T|$ versus frequency.

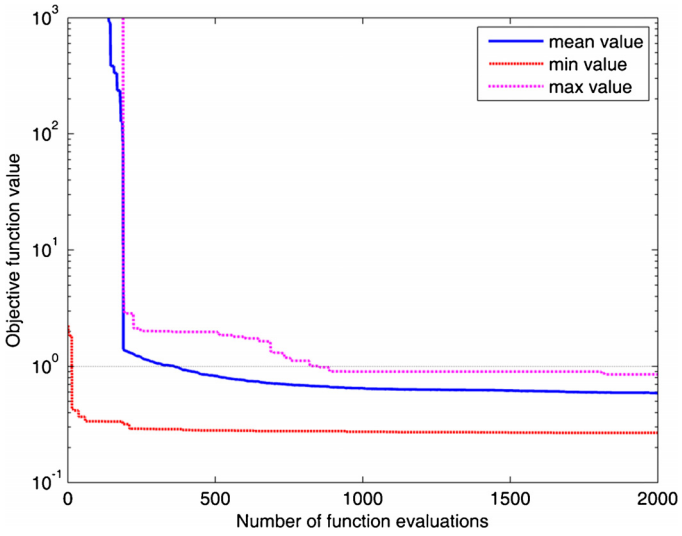

Fig. 6. Value of the objective function given in (14) versus the number of FE's. The results are averaged over 100 independent runs.

of process plus the weights w_I and w_P [30], the transfer function of controller can be considered as the following

$$K(s) = \frac{b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0}, \tag{16}$$

where $a_i, b_i \in \mathbb{R}$ $(i = 0, 1, 2)$ are unknown parameters of the controller to be determined by solving (14) (note that here we have $\mathbf{x} = [a_0\, a_1\, a_2\, b_0\, b_1\, b_2]$). Assuming $\lambda = 10^5$, $N_{pop} = 10$, $d_{runner} = 10^{10}$, $d_{root} = 10^8$, $a = 0.1$, $stall\_max = 2000$, $tol = 10^{-3}$ and $\mathbf{x}_u = -\mathbf{x}_l = 10^{10}$, and after few independent runs (each stops after 2000 FE's) the solution of (14) using RRA is obtained as

$$K(s) = \frac{2.504 \times 10^8 s^2 + 1.679 \times 10^9 s + 1 \times 10^{10}}{s^3 + 1.922 \times 10^6 s^2 + 1 \times 10^{10} s + 1 \times 10^{10}}, \tag{17}$$

for which we have $\gamma = \left\| |w_P S| + |w_I T| \right\|_\infty = 0.2879$ and $g(\mathbf{x}^*) = -3.2364$. Note that since the solutions that lead to unstable feedback systems are quite useless in practice, the value of λ is considered very large to make sure that such solutions are abandoned rapidly by algorithm. Moreover, no re-initialization occurs in this problem since each run stops after 2000 FE's which is equal to the value assigned to *stall_max*. Fig. 5 shows $|w_P S| + |w_I T|$ versus frequency (in log scale) for the controller given in (17). As it is observed, the plot is under 0 dB at all frequencies which guarantees NP, RS, and RP (the NS is achieved by the negative value obtained for $g(\mathbf{x}^*)$ in log scale).

Fig. 6 shows the objective function value (as defined in (14)) versus the number of FE's in this problem (the simulations are performed assuming $\lambda = 10^5$, $N_{pop} = 10$, $d_{runner} = 10^{10}$, $d_{root} = 10^8$, $a = 0.1$, $stall\_max = 2000$, $tol = 10^{-3}$ and the results are averaged over 100 independent runs). It is concluded from this figure that the average value of objective function becomes less than unity after 362 FE's, which means that it is expected that the algorithm can find a

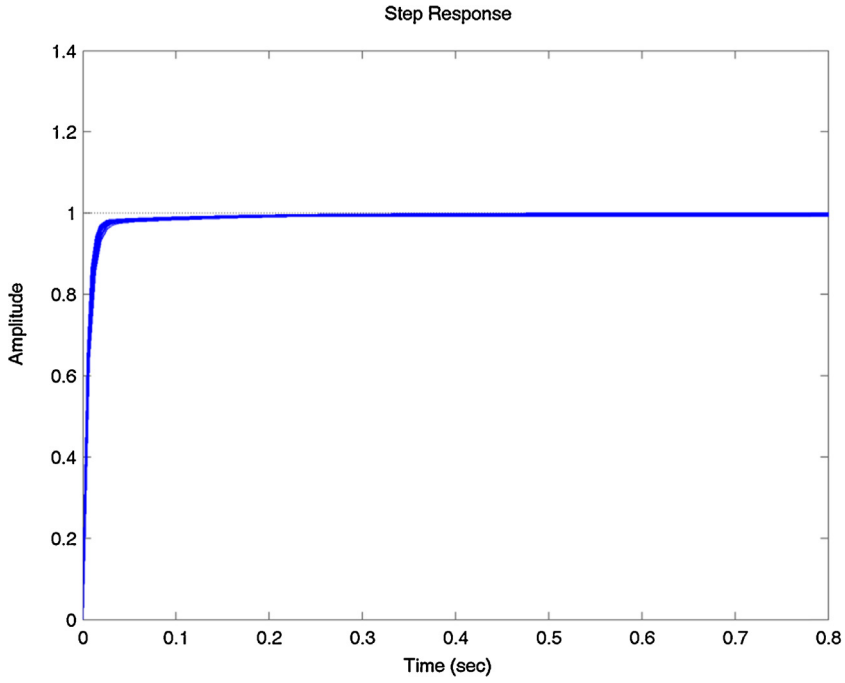

Fig. 7. Unit step response of the feedback system for 30 different random uncertainties.

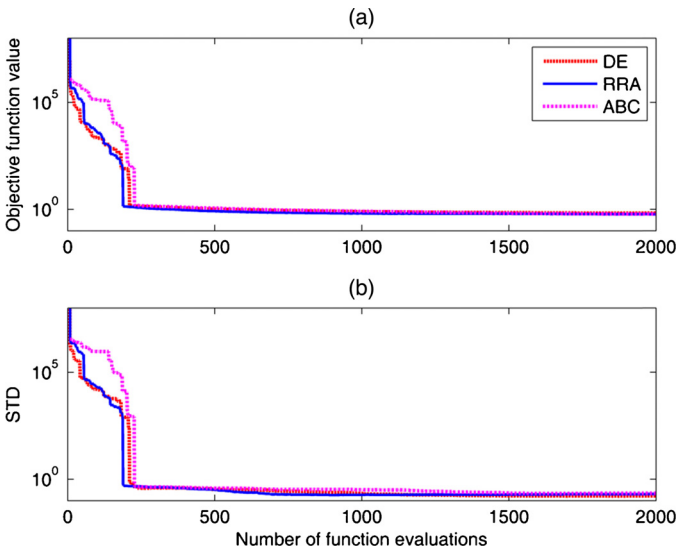

Fig. 8. Comparing RRA with DE and ABC, (a) objective function value (averaged over 100 independent runs), (b) corresponding standard deviation.

solution for this problem after such number of FE's. Fig. 7 shows the unit step response of the closed-loop system under consideration for 30 different random Δ's when the controller given in (17) is used.

In order to convince the readers about the good performance of RRA for solving this special problem, the results are also compared with those obtained by using state-of-the-art algorithms DE (the DE/rand/1/bin with per-generation-dither variant) and ABC (the version presented in [36]). For problem solving using DE the number of population members, differential weighting factor (F), and crossover probability constant (CR) are considered equal to 10, 0.3, and 0.5, respectively. When using ABC, the number of colony size and *limit* are considered equal to 14 and 200, respectively. The parameters of RRA are also considered exactly equal to those used in simulations of Fig. 6. Each algorithm is independently run 100 times and each time it stops after 2000 FE's. Note that the control parameters of each algorithm are selected by trial and error such that it exhibits the best performance. The average value of objective function (14) and the corresponding standard deviation are plotted versus the number of FE's in Fig. 8(a) and (b), respectively (note that in order to make a fair comparison it is always better to plot the objective function value versus the number of FE's [37–39]). As it can be observed in this figure the performance of RRA is comparable with ABC and DE.

6. Conclusion

A numerical optimization algorithm inspired by the function of runners and roots of some plants in nature, the runner-root algorithm (RRA), is presented in this paper. The proposed algorithm first searches the domain of problem by using random large steps, which are called runners. This procedure continues until no considerable improvement is observed in the best solution, which causes the beginning of local search procedure through roots and root hairs. The algorithm also restarts automatically if no considerable improvement is observed in the value of cost function after certain number of successive iterations.

The performance of RRA is compared (through CEC'2005 benchmark problems) with 9 state-of-the-art metaheuristic algorithms using nonparametric statistical methods including Wilcoxon signed rank test (for pairwise comparisons) and Friedman test (for multiple comparisons). The pairwise comparisons results obtained by using Wilcoxon signed rank test followed by Holm's post hoc procedure showed the improvement of RRA over CHC, SS-Arit, SS-BLX, PSO, and SSGA with the level of significance $\alpha = 0.05$. However, no improvement was detected in pairwise comparison of RRA with IPOP-CMA-ES, DE-Exp, SaDE, and DE-Bin. The $1 \times N$ multiple comparisons performed by the standard Friedman test showed that RRA stands at the fourth rank (after DE-Exp, DE-Bin, and SaDE). After calculating the corresponding adjusted p-values using Holm's post hoc procedure (assuming RRA as the control method) significant improvement of RRA over PSO and CHC with the level of significance $\alpha = 0.05$ was detected. However, no improvement of RRA over the remaining algorithms was detected using Friedman test followed by Holm's post hoc procedure.

Sensitivity analysis on control parameters of RRA is also briefly studied. The sensitivity analysis performed on one of the benchmark problems (where RRA outperformed others with nominal values for control parameters) revealed the fact that RRA preserves its superiority even after subjecting the control parameters to some changes around their nominal values. The sensitivity analysis on other benchmark problem showed that the performance of algorithm can be improved by more appropriate choice of its control parameters. RRA was also successfully used to solve an open problem in the field of robust control theory and the results were compared with DE/rand/1/bin (with per-generation-dither) and ABC. These comparisons showed that the performance of RRA is comparable with these two state-of-the-art algorithms in dealing with this special problem.

The current version of RRA applies an adaptive method to control the ratio between global (exploration) and local searches (exploitation). Developing a version of this algorithm with self-adaptation capacity can be considered as the subject of future studies.

Acknowledgment

The author is indebted to F. Hojjati-Parast for pointing out the role of roots in plants as tools for local search for water resources and minerals. The author also gratefully thanks anonymous reviewers for suggesting many useful comments to improve the quality of this work.

References

[1] I. Rechenberg, Cybernetic solution path of an experimental problem, in: Royal Aircraft Establishment Translation No. 1122, B.F. Toms, Trans, 1965.
[2] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, University of Michigan Press, Oxford, England, 1975.
[3] R.C. Eberhart, Y. Shi, Particle swarm optimization: developments, applications and resources, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, 27–30 May 2001, Seoul, 2001, pp. 81–86.
[4] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IV IEEE International Conference on Neural Networks, Piscataway, New Jersey, 1995, pp. 1942–1948.
[5] M. Dorigo, M. Birattari, Ant colony optimization, in: C. Sammut, I.G. Webb (Eds.), Encyclopedia of Machine Learning, Springer, 2010, pp. 36–39.
[6] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39.
[7] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (1) (2008) 687–697.
[8] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Glob. Optim. 39 (3) (2007) 459–471.
[9] V. Cerny, A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, J. Optim. Theory Appl. 45 (1985) 41–51.
[10] S. Kirkpatrick, C.D. Gellat, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[11] R.V. Rao, V.J. Savsani, D.P. Vakharia, Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems, Comput. Aided Des. 43 (3) (2011) 303–315.
[12] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. Mag. 22 (3) (2002) 52–67.

[13] L.N. De Castro, J. Timmis, Artificial Immune Systems: A New Computational Intelligence Approach, Springer, 2002, ISBN 978-1-85233-594-6.
[14] M.M. Eusuff, K.E. Lansey, Optimization of water distribution network design using the shuffled frog leaping algorithm, J. Water Resour. Plan. Manag. 129 (2003) 210–225.
[15] K.V. Price, M. Rainer, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer-Verlag, 2005.
[16] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 2, 2005, pp. 1785–1791.
[17] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in: Proceedings of the IEEE Congress on Evolutionary Computation, 25–28 September, Singapore, 2007, pp. 4661–4667.
[18] Y. del Valle, G.K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, R.G. Harley, Particle swarm optimization: basic concepts, variants and applications in power systems, IEEE Trans. Evol. Comput. 12 (2) (2008) 171–195.
[19] M.R. AlRashidi, M.E. El-Hawary, A survey of particle swarm optimization applications in electric power systems, IEEE Trans. Evol. Comput. 13 (4) (2009) 913–918.
[20] P.J. Fleming, R.C. Purshouse, Evolutionary algorithms in control systems engineering: a survey, Control Eng. Pract. 10 (11) (2002) 1223–1241.
[21] A. Salhi, E.S. Fraga, Nature-inspired optimisation approaches and the new plant propagation algorithm, in: Proceedings of the International Conference on Numerical Analysis and Optimization (ICeMATH 2011), vol. 1, Yogyakarta, Indonesia, 2011, pp. 1–8.
[22] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC'2005 special session on real parameter optimization, Nanyang Technological University, Tech. Rep, 2005, http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC-05/CEC05.htm
[23] G.W. Corder, D.I. Foreman, Nonparametric Statistics: A Step-by-Step Approach, Wiley, 2014.
[24] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (2011) 3–18.
[25] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[26] C.A. Coello Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Comput. Methods Appl. Mech. Eng. 191 (11–12) (2002) 1245–1287.
[27] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. 45 (3) (2013) 1–33, Article No. 35.
[28] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 2005, pp. 1769–1776.
[29] L.J. Eshelman, J.D. Schaffer, Real-coded genetic algorithms and interval schemata, in: D. Whitley (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1993, pp. 187–202.
[30] C. Fernandes, A. Rosa, A study of non-random matching and varying population size in genetic algorithm using a royal road function, in: Proceedings of the 2001 Congress on Evolutionary Computation, Piscataway, New Jersey, 2001, pp. 60–66.
[31] H. Mülenbein, D. Schlierkamp-Voosen, Predictive models for the breeding genetic algorithm in continuous parameter optimization, Evol. Comput. 1 (1993) 25–49.
[32] M. Laguna, R. Martí, Scatter Search. Methodology and Implementation in C, Kluwer Academic Publishers, 2003.
[33] F. Herrera, M. Lozano, D. Molina, Continuous scatter search: an analysis of the integration of some combination methods and improvement strategies, Eur. J. Oper. Res. 169 (2) (2006) 450–476.
[34] S.-H. Liu, M. Mernik, D. Hrnčič, M. Črepinšek, A parameter control method of evolutionary algorithms using exploration and exploitation measures with a practical application for fitting Sovova's mass transfer model, Appl. Soft Comput. 13 (2013) 3792–3805.
[35] S. Skogestad, I. Postlethwaite, Multivariable Feedback Control, 2nd ed., John Wiley & Sons, UK, 2005.
[36] D. Karaboga, B. Akay, A comparative study of Artificial Bee Colony algorithm, Appl. Math. Comput. 214 (2009) 108–132.
[37] M. Črepinšek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (2014) 161–170.
[38] M. Mernik, S.-H. Liu, D. Karaboga, M. Črepinšek, On clarifying misconceptions when comparing variants of the Artificial Bee Colony Algorithm by offering a new implementation, Inf. Sci. 291 (2015) 115–127.
[39] M. Črepinšek, S.-H. Liu, L. Mernik, A note on teaching–learning-based optimization algorithm, Inf. Sci. 212 (2012) 79–93.