Computers and Mathematics with Applications 60 (2010) 2087–2098

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search

R. Oftadeh *, M.J. Mahjoob, M. Shariatpanahi

Center for Mechatronics and Automation, School of Mechanical Engineering, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Received 17 September 2009
Received in revised form 23 June 2010
Accepted 27 July 2010

Keywords:
Meta-heuristic algorithm
Continuous optimization problems
Group hunting

ABSTRACT

A novel optimization algorithm is presented, inspired by group hunting of animals such as lions, wolves, and dolphins. Although these hunters have differences in the way of hunting, they are common in that all of them look for a prey in a group. The hunters encircle the prey and gradually tighten the ring of siege until they catch the prey. In addition, each member of the group corrects its position based on its own position and the position of other members. If the prey escapes from the ring, hunters reorganize the group to siege the prey again. Several benchmark numerical optimization problems, constrained and unconstrained, are presented here to demonstrate the effectiveness and robustness of the proposed Hunting Search (HuS) algorithm. The results indicate that the proposed method is a powerful search and optimization technique. It yields better solutions compared to those obtained by some current algorithms when applied to continuous problems.

1. Introduction

Classical methods often face great difficulties in solving optimization problems that abound in the real world. In order to overcome the shortcomings of traditional mathematical techniques, nature-inspired soft computing algorithms have been introduced.

Several evolutionary or meta-heuristic algorithms have since been developed which combine rules and randomness mimicking natural phenomena. These phenomena include biological evolutionary processes (e.g., the evolutionary algorithm proposed by Fogel et al. [1], De Jong [2], and Koza [3] and the genetic algorithm (GA) proposed by Holland [4] and Goldberg [5]), animal behavior (e.g., the tabu search proposed by Glover [6]), the physical annealing process (e.g., simulated annealing proposed by Kirkpatrick et al. [7]) and the musical process of searching for a perfect state of harmony (proposed by Geem et al. [8], Lee and Geem [9] and Geem [10] and proceeded with other researchers [11,12]).

Many researchers have recently studied these meta-heuristic algorithms, especially GA-based methods, to solve various optimization problems. However, new heuristic algorithms are needed to solve difficult and complicated real-world problems.

The method introduced in this paper is a meta-heuristic algorithm which simulates the behavior of animals hunting in a group (lions, wolves, etc.). Group hunters have certain strategies to encircle the prey and catch it. Wolves, for instance, rely on this kind of hunt very much, so they can hunt animals bigger or faster than themselves. They choose one prey and the group gradually moves toward it. They do not stand in the wind such that the prey senses their smell. We employ this idea in constrained problems to avoid forbidden areas. In our algorithm, each of the hunters indicates one solution for a particular problem. Like real animals which hunt in a group, artificial hunters cooperate to find and catch the prey; i.e., the optimum point in our work.

* Corresponding author.
E-mail addresses: roftadeh@me.ut.ac.ir, roftadeh@ut.ac.ir (R. Oftadeh).

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.07.049

2. Hunting search meta-heuristic algorithm

Meta-heuristic algorithms imitate natural phenomena, e.g. physical annealing in simulated annealing, human memory in a tabu search, and evolution in evolutionary algorithms. A new Hunting Search (HuS) meta-heuristic algorithm is conceptualized here using the hunt process in catching a prey in the group hunting. Cooperation of the members of the group called hunters leads to encircling a prey and catching it (the group's food), similar to the optimization process which results in finding a global solution (a perfect state) as determined by an objective function. The position of each hunter compared to the prey determines its chance of catching the prey. Similarly, the objective function value is determined by the set of values assigned to each decision variable. The new HuS meta-heuristic algorithm is derived based on a model of group hunting of animals when searching for food such as the way in which wolves hunt.

In continuous optimization problems, the estimation of a solution is carried out by putting values of decision variables into the objective function or fitness function. This evaluates the function value, which includes cost, efficiency, and/or error. Therefore, a model for a continuous optimization problem may be formally defined as follows.

Definition 2.1. A model of a continuous optimization problem, generally shown as $Q = (\boldsymbol{S}, \boldsymbol{\Omega}, f)$, has the following features.

- A search space defined over a finite set of continuous decision variables ($\boldsymbol{S}$).
- A set of constraints among the variables ($\boldsymbol{\Omega}$).
- An objective function to be minimized ($f : \boldsymbol{S} \to \mathbb{R}_{\mathbf{0}}^{+}$).

The search space ($\boldsymbol{S}$) is defined as a set of N continuous variables ($x_i,\ i = 1, \ldots, N$ with values $v_i \in \boldsymbol{D}_i \subseteq \mathbb{R}$), where N is the number of design variables. A solution $s \in \boldsymbol{S}$ which satisfies all the constraints in the set $\boldsymbol{\Omega}$ is a feasible solution. Q is called an unconstrained problem if the set $\boldsymbol{\Omega}$ is empty; otherwise, it is called a constrained problem. A solution $s^* \in \boldsymbol{S}$ is called a global minimum if and only if $f(s^*) \leq f(s)\ \forall s \in \boldsymbol{S}$. Solving a continuous optimization problem requires at least one global minimum.

Compared to group hunting, in a continuous optimization problem each 'hunter' is replaced with a 'solution' of the problem (or as we call it an 'artificial hunter'). Note that group hunting of animals and our meta-heuristic algorithm have a primary difference. In group hunting of animals (our emphasis is on animals that hunt on land such as wolves and lions), hunters can see the prey or when they hunt at night at least they can sense the smell of the prey and determine its position. In contrast, in optimization problems we have no indication of the optimum solution/point. In group hunting of animals, however, the solution (prey) is dynamic and the hunters (based on the current position of the prey) must correct their position. In optimization problems instead, the optimum solution is static and does not change its position during the search process. In fact, both real and artificial group hunting have their own difficulties. To resemble this dynamics of the hunting process in our algorithm, artificial hunters move towards the leader. The leader is the hunter which has the best position at the current stage (the optimum solution among current solutions at hand). In fact, we assume that the leader has found the optimum point and other members move towards it. If any of them finds a point better than the current leader, it becomes leader in the next stage.

Real animals not only gradually move toward the prey but also (based on the position of other hunters and the position of the prey) correct their position. Therefore, in this algorithm, after moving toward the previous leader, the hunters correct their position based on the position of other members. This is accomplished by introducing the 'hunting group consideration rate' (HGCR), which is defined later.

In addition in the group hunting of real animals, if the prey escapes out of the ring, the hunters organize themselves to encircle the prey again. In the HuS algorithm, the ability will be given to hunters, so they can search out of the ring of siege.

In the algorithm, if the positions of the hunters/solutions are too close to each other, the group is reorganized to find the optimum point in the next effort.

Fig. 1 displays the procedure of the Hunting Search algorithm, which consists of the following steps.

Step 1. Specify the optimization problem and parameters of the algorithm.
Step 2. Initialize the hunting group (HG).
Step 3. Move toward the leader.
Step 4. Correct the positions (cooperation between members).
Step 5. Reorganize the hunting group.
Step 6. Repeat Steps 3, 4 and 5 until the termination criterion is satisfied.

The details follow.

Step 1. *Initialize the optimization problem and algorithm parameters.*
The problem is defined as the model that is presented in Definition 2.1. The HuS algorithm parameters that are required to solve the optimization problem are also specified in this step: hunting group size (number of solution vectors in hunting group HGS), maximum movement toward the leader (MML), and hunting group consideration rate (HGCR), which varies between 0 and 1. The parameters MML and HGCR are parameters that are used to improvise the hunter position (solution vector) that are defined in Steps 3 and 4.

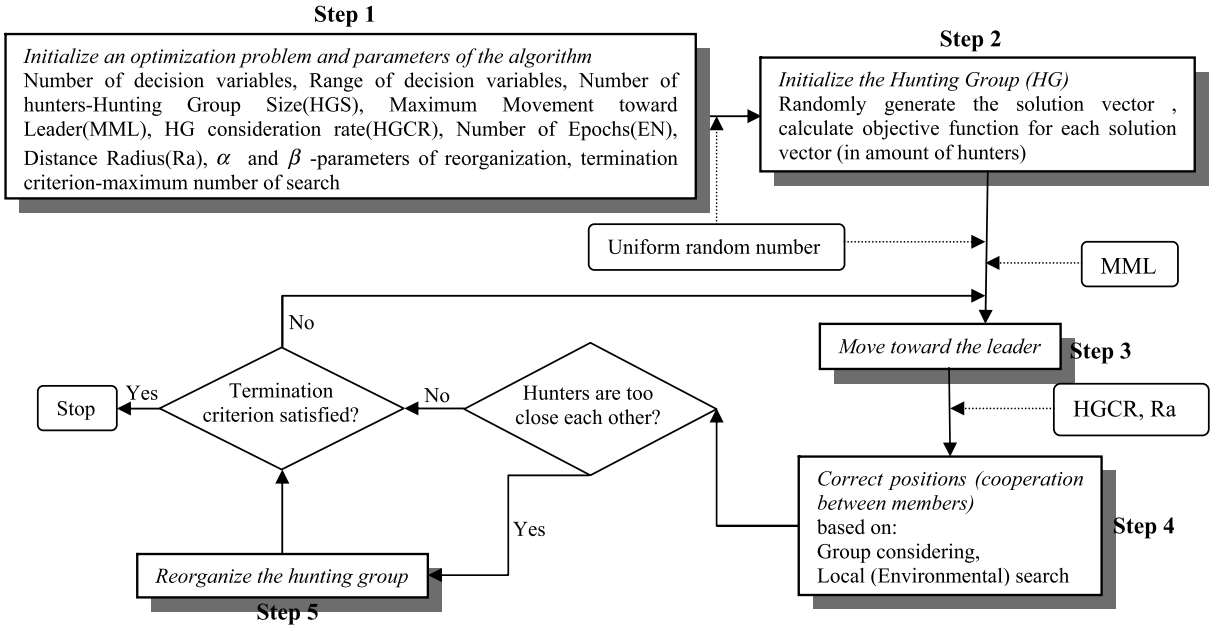

Fig. 1. Optimization procedure of the Hunting Search algorithm.

Step 2. *Initialize the hunting group (HG).*
Based on the number of hunters (HGS), the hunting group matrix is filled with feasible randomly generated solution vectors. The values of objective function are computed and the leader is defined based on the values of objective functions of the hunters.
Step 3. *Moving toward the leader.*
The new hunters' positions (new solution vectors) $x' = (x'_1, x'_2, \ldots, x'_N)$ are generated by moving toward the leader (the hunter that has the best position in the group) as follows:

$$x'_i = x_i + \text{rand} \times \text{MML} \times (x_i^L - x_i). \tag{1}$$

The MML is the maximum movement toward the leader, rand is a uniform random number which varies between 0 and 1, and x_i^L is the position value of the leader for the ith variable.

For each hunter, if the movement toward the leader is successful, the hunter stays in its new position. However, if the movement is not successful (its previous position is better than its new position) it comes back to the previous position. With this, we achieve some advantages. First, we do not compare the hunter with the worst hunter in the group, so we allow the weak members to search for other solutions; they may find better solutions (as in a genetic algorithm). Secondly, for prevention from rapid convergence of the group the hunter compares its current position with its previous position; therefore, good positions will not be eliminated.

The value of parameter MML varies for different problems. It depends on the number of iterations in each *epoch*, which will be defined in Step 5. The range between 0.05 (for epochs with large number of iterations) and 0.4 (for epochs with small number of iterations) gives the best results, as is shown in the next examples.

Step 4. *Position correction—cooperation between members.*
In this step, the cooperation between hunters is modeled in order to conduct the 'hunt' more efficiently. After moving toward the leader, hunters (based on other hunter positions and some random factors) choose another position to find better solutions. Hunters do the position correction in two ways: (1) real value correction (2) digital value correction.
In real value correction, the new hunter's position $x' = (x'_1, x'_2, \ldots, x'_N)$ is generated from HG, based on hunting group considerations or position corrections. For instance, the value of the first design variable for the jth hunter ($x_1^{j'}$) for the new vector (new hunter position) can be picked from any value (i.e. real number) in the specified HG ($x_i^1, x_i^2, \ldots, x_i^{\text{HGS}}$) or corrected using the HGCR parameter (chosen between 0 and 1). Updating the variable value is thus carried out as follows:

$$x_i^{j'} \leftarrow \begin{cases} x_i^{j'} \in \{x_i^1, x_i^2, \ldots, x_i^{\text{HGS}}\} & \text{with probability HGCR} \\ x_i^{j'} = x_i^j \pm \text{Ra} & \text{with probability } (1 - \text{HGCR}) \end{cases} \quad i = 1, \ldots, N,\ j = 1, \ldots, \text{HGS}. \tag{2}$$

The parameter HGCR is the probability of choosing one value from the hunting group stored in the HG, and (1 − HGCR) is the probability of doing a position correction. For example, an HGCR of 0.3 indicates that the HuS algorithm will choose the

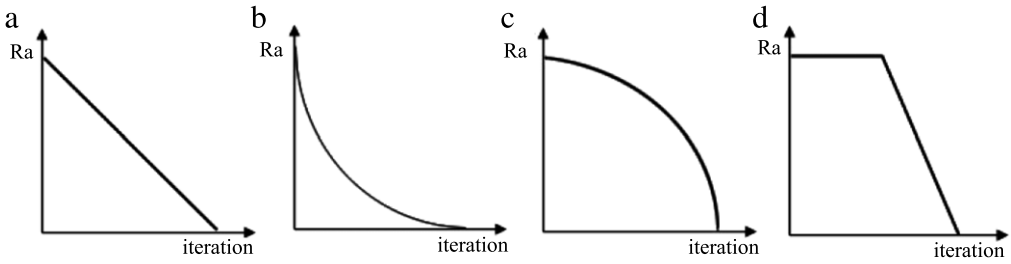

Fig. 2. Ra reduction: (a) constant reduction; (b), (c) exponential decay; (d) Ra begins to reduce after a certain number of searches.

design variable value from hunting group values in the HG with a 30% probability, and doing a position correction with a 70% probability. An HGCR in the range between 0.1 and 0.4 gives better results, as is shown in the examples.
Ra is an arbitrary distance radius for the continuous design variable, and rand is a uniform number which varies between 0 and 1. Ra can be fixed or reduced during optimization process. Many functions can be chosen for reducing Ra. Some useful functions are plotted in Fig. 2. The first function uses a fixed decremental amount, which is subtracted from the search Ra after each iteration. This results in a constant rate of Ra reduction. The second function uses a fixed Ra reduction factor (some value less than 1), by which the Ra is multiplied after each generation. This option allows hunters to narrow down their search more rapidly initially, and to spend longer in detailed local solution exploration. The third function uses a similar approach, but instead is more biased towards global exploration. This approach leaves the hunters with more time to investigate the properties of the entire search space, and undergoes rapid convergence on solutions towards the end of the algorithm's life. In the fourth function, the Ra only begins to contract after a certain number of iterations.
In this paper, an exponential function is used for Ra reduction as follows:

$$\mathrm{Ra}(it) = \mathrm{Ra}_{\min}(\max(x_i) - \min(x_i)) \exp\left(\frac{\mathrm{Ln}\left(\frac{\mathrm{Ra}_{\min}}{\mathrm{Ra}_{\max}}\right) \times it}{itm}\right), \tag{3}$$

where it is the iteration number, $\max(x_i)$ and $\min(x_i)$ are the maximum or minimum possible value of variable x_i, respectively, $\mathrm{Ra}_{\max}$ and $\mathrm{Ra}_{\min}$ are the maximum and minimum of relative search radius of the hunter, respectively, and itm is the maximum number of iterations in the optimization process.
In digital value correction, instead of using real values of each variable, the hunters communicate with each other by the digits of each solution variable. For example, the solution variable with the value of 23.4356 has six meaningful digits. For this solution variable, the hunter chooses a value for the first digit (i.e. 2) based on hunting group considerations or position correction. For HGCR = 0.3, it chooses a digit from the hunting group with a 30% probability and does a position correction with 70% probability. The values for other digits in this solution variable and other variables can be selected in the same manner. Therefore, the kth digit of the ith solution variable is chosen as follows:

$$d_{ik}^{j'} \leftarrow \begin{cases} d_{ik}^{j'} \in \{d_{ik}^1, d_{ik}^2, \ldots, d_{ik}^{\mathrm{HGS}}\} & \text{with probability HGCR} \\ d_{ik}^{j'} = d_{ik}^{j} \pm a & \text{with probability } (1 - \mathrm{HGCR}) \end{cases} \quad \begin{array}{l} i = 1, \ldots, N,\ j = 1, \ldots, \mathrm{HGS}, \\ k = 1, \ldots, M \text{ (number of digits in each variable)}. \end{array} \tag{4}$$

The value of a can be any number between 1 and 9 or a random variable that generates numbers between 1 and 9. In the numerical examples we set the value of a to 1.
There are some points that should be considered in digital position correction. In the group consideration, there is a chance that a digit chooses another digit that is invalid (or void) and makes the value of solution variable inadmissible. For example, if the second digit of 423.3423 chooses the similar digit in 2.5678 that is void, this makes the value of solution variable inadmissible. Also, in the position correction, if the kth digit of solution variable is 9 or 0 and then added to or decreased by 1, this makes the value of the solution variable inadmissible (10 or −1). In such cases, the digit simply restores its original value.
Note that in digital position correction the algorithm must restore a certain number of digits for each solution variable. In the present study, for all numerical examples, the algorithm restores up to eight decimal figures.
The algorithm can use real value correction, digital value correction or both for cooperation between members. We have used both here. The presented numerical examples showed the effectiveness of this choice.
After the quality of the new hunter position is determined by evaluating the objective function, the hunter moves to this new position; otherwise it keeps its previous position (similar to the previous step).

Step 5. *Reorganizing the hunting group*.
As the search process continues, there is a chance for the hunters to be trapped in a local minimum (or a local maximum once our goal is to find the maximum). If this happens, the hunters must reorganize themselves to get another opportunity to find the optimum point. The algorithm does this in two independent conditions. If the difference between the values of the objective function for the leader and the worst hunter in the group becomes smaller than a preset constant (ϵ_1) and

Table 1
HuS parameter values for constrained minimization problem Eq. (6).

Parameter	Value
Number of epochs (NE)	2
Iteration per epoch (IE)	30
Hunting group size (HGS)	10
Maximum movement toward leader (MML)	0.3
Hunting group consideration rate (HGCR)	0.3
Ra_{max}, Ra_{min}	1e−2, 1e−7
Reorganization parameters $-\alpha$, β	0.1, −1

the termination criterion is not satisfied, then the algorithm reorganizes the hunting group for each hunter. Alternatively, after a certain number of searches the hunters reorganize themselves. We have used the second condition in the examples for reorganization. The sequence of searches that end with trapping the group in a local minimum or the certain number of searches is called one *epoch*. They reorganize as follows. The leader keeps its position and the other hunters randomly choose their positions in the design space by

$$x_i' = x_i^L \pm \text{rand} \times (\max(x_i) - \min(x_i)) \times \alpha \exp(-\beta \times \text{EN}), \tag{5}$$

where x_i^L is the position value of the leader for the ith variable. rand is a uniform random number which varies between 0 and 1. $\max(x_i)$ and $\min(x_i)$ are the maximum and minimum possible values of variable x_i, respectively. EN counts the number of times that the group has been trapped until this step (i.e. number of epochs until this step). As the algorithm goes on, the solution gradually converges to the optimum point. Parameters α and β are positive real values. They determine the global convergence rate of the algorithm. Large values of α and small values of β cause the algorithm to converge slowly. This is recommended for large optimization problems or for problems with several local optimums. In contrast, setting small values of α and large values of β makes the algorithm converge more rapidly. This is recommended for small optimization problems with a small number of design variables.

By this practice, hunters benefits in four ways. First, they give themselves another opportunity to search the design space. Second, because the leader saves its position, other hunters have a direction to better areas after reorganizing and they do not have to search the whole design space blindly. Third, because the leader saves its position after each epoch, hunters do not miss the best solution they have found during search process. Fourth, and most importantly, as EN increases during search process, the hunters search the design space more locally to find the optimum point.

As the algorithm proceeds, the solution gradually converges into the optimum point.

Step 6. *Repeat Steps* 3–5 *until the termination criterion is satisfied*.

In Step 6, the computations are terminated when the termination criterion is satisfied. If not, Steps 3–5 are then repeated. The termination criterion can be defined as the maximum number of searches. Alternatively, if after reorganizing the function for the leader and the worst hunter in the group remains smaller than a preset constant (ϵ_1), the search process ends.

To further elaborate on the HuS meta-heuristic algorithm, consider the following constrained minimization problem:

$$\begin{aligned} &f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\ &\text{subject to} \\ &g_1(x) = 4.84 - (x_1 - 0.05) - (x_2 - 2.5)^2 \geq 0 \\ &g_2(x) = x_1^2 + (x_2 - 2.5)^2 - 4.84 \geq 0 \\ &0 \leq x_1 \leq 6, \qquad 0 \leq x_2 \leq 6. \end{aligned} \tag{6}$$

This problem, introduced by Deb [13], is a minimization problem with two design variables and two inequality constraints, as shown in Fig. 3. The unconstrained objective function $f(x)$ has a minimum solution at $(3, 2)$ with a corresponding function value equal to 0. The constrained minimum solution is located in a narrow crescent-shaped region. The HuS parameter values used in this problem are summarized in Table 1 (step 1).

The HG was initially structured with randomly generated solution vectors within the bounds prescribed for this example (i.e., 0 to 6.0) and the leader is defined (Step 2). Next, based on Eq. (1), the hunters move toward the leader, and if their new positions are better than the previous positions, they stay there. Otherwise, they come back to their previous positions (Step 3).

Then the hunters cooperate with each other to find better positions based on group considerations with a 30% probability and position corrections with a 70% probability (Step 4). After 30 iterations based on Eq. (6), the hunters reorganize themselves (Step 5). In this example, the group can find the minimum point in two epochs. After 60 iterations (1800 function evaluations), the HuS algorithm improvised an optimal solution vector $x = (2.246826212191825\ 2.381870437963585)$, which has a function value of 13.590841693489994, as shown in Fig. 4.

Fig. 4 shows the minimum value convergence history for this constrained function. The HuS best solution was compared to the previous solutions reported in the literature in Table 2. The optimal design obtained using the HuS algorithm showed a very good agreement with the previous solutions reported in the literature.

Table 2
Optimal results of the constrained function represented by Eq. (5).

Methods	Optimal design variables (x)		Constraints		Objective function value $f(x)$
	x_1	x_2	g_1	g_2	
Deb [13]	–	–	–	–	13.59085
Lee and Geem [9]	2.246840	2.382136	0.00002	0.22218	13.590845
Mahdavi [11]	2.2468258	2.381863	0.0000	0.22218	13.590841
Present study	**2.2468262121**	**2. 3818704379**	**2.21003** $\times$ $\mathbf{10^{-15}}$	**0.2221826212**	**13.5908416934**

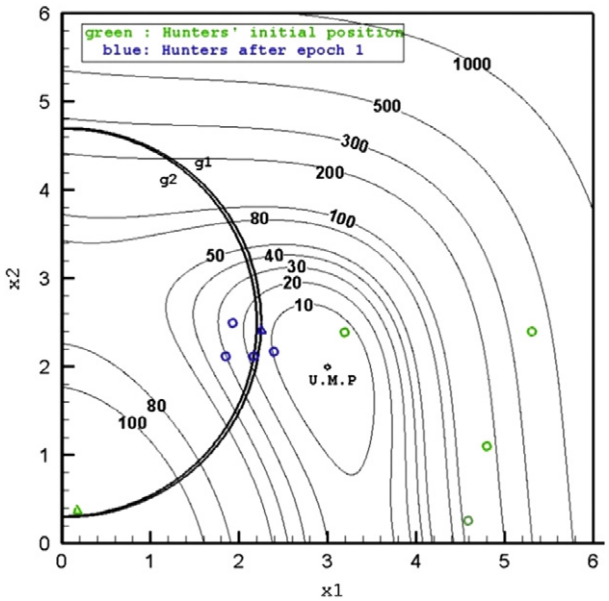

Fig. 3. Constrained function represented by Eq. (5). Green circles indicate hunters' initial positions. Blue circles indicate hunters' positions after epoch 1. The leader in each of these stages is represented by a triangle. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

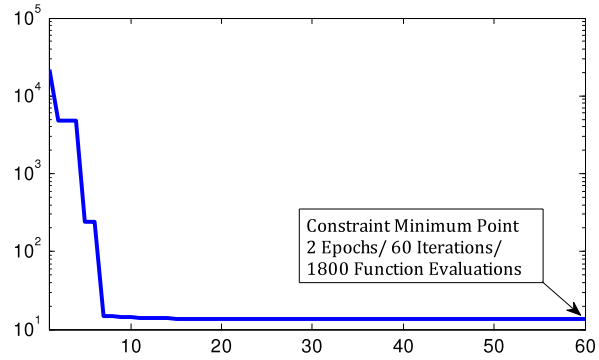

Fig. 4. Minimum value convergence history for the constrained function represented by Eq. (5).

The HuS algorithm includes the structure of present meta-heuristic optimization algorithms. It preserves the history of past vectors (HG) similar to the tabu search and the harmony search, and is able to vary the adaptation rate (HGCR) from the beginning to the end of the computations, which resembles simulated annealing. It also considers several vectors simultaneously in a manner similar to the harmony search. However, the difference between the harmony search and the developed HuS algorithm is that in the latter hunters compare their positions only with their previous positions; therefore weak hunters have a chance to find better locations in subsequent iterations, while the harmony search compares each generated solution with all existing solutions and removes the worst solution from harmony memory.

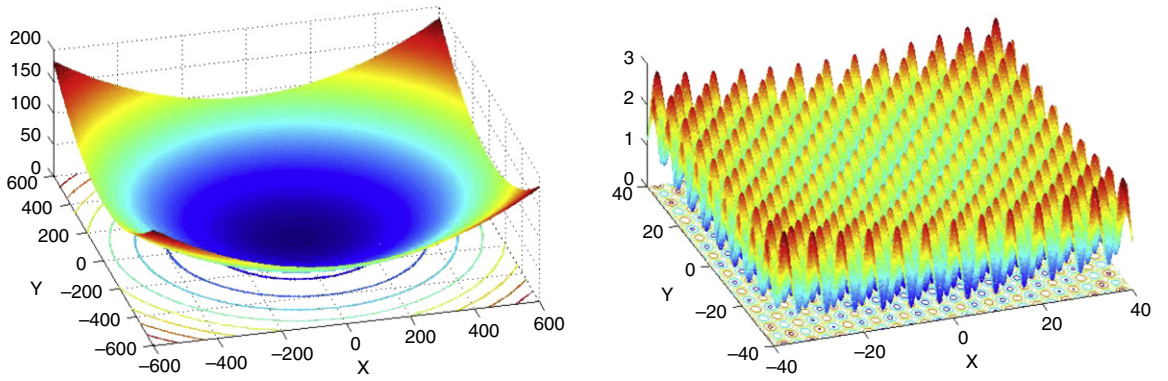

Fig. 5. Schematic of a Griewank function with different resolutions represented by Eq. (7). Left: the full definition range of the function. Right: range around the optimum point.

Table 3
HuS parameter values for the Griewank function Eq. (7).

Parameter	Value
Problem dimension (d)	10, 20, 50, 100
Number of epochs (NE)	3 ($d = 10, 20$), 5 ($d = 50$), 10 ($d = 100$)
Iteration per epoch (IE)	50
Hunting group size (HGS)	10
Maximum movement toward leader (MML)	0.3
Hunting group consideration rate (HGCR)	0.3
$\mathrm{Ra}_{\max}$, $\mathrm{Ra}_{\min}$	10^{-2}, 5×10^{-6}
Reorganization parameters $-\alpha$, β	0.05, −0.5

3. Examples

The computational procedures described above have been implemented in a MATLAB computer program on a Pentium4 2.4 GHz computer. In this study, different unconstrained and constrained standard benchmark examples from the literature are presented to demonstrate the efficiency and robustness of the proposed HuS meta-heuristic algorithm. Unconstrained optimization problems are presented in two sets. In these sets, the performance of HuS is compared with five famous evolutionary algorithms through minimization of a high-dimensional Griewank function and other standard benchmark problems available in the literature. Three constrained optimization problems are also included to test the efficiency of the presented algorithm in the optimization of this type of problem.

3.1. Unconstrained function minimization examples—set one

In this section, the performance of HuS is compared with that of four famous evolutionary algorithms. Genetic Algorithms (GAs) are inspired by biological system's improved fitness through evolution [4]. Memetic Algorithms (MAs) are inspired by Dawkins' notion of a meme [14]. Particle Swarm Optimization (PSO) is inspired by the social behavior of a flock of migrating birds trying to reach an unknown destination [15]. The Shuffled Frog Leaping (SFL) algorithm is inspired by a frog population searching for food and passing their ideas through a shuffling process [16]. The results are compared with those reported in [17].

To have a reasonable comparison between HuS and the mentioned optimization algorithms, the simulations are performed for a high-dimension Griewank function ($d = 10, 20, 50$, and 100). The function is represented by

$$f_{\mathrm{GR}}(\vec{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad -511 \le x_i \le 512 \ (d = 10, 20, 50, 100). \tag{7}$$

A schematic of a two-dimensional Griewank function is given in Fig. 5. As can be observed, this function has many widespread local minima but it has only one global minimum at $x_i = 0,\ i = 1 : n$ with $f(x) = 0$. The HuS parameter values used in this problem are summarized in Table 3.

As in [17], 20 trial runs were performed for each dimension with the two (following) stopping criteria: (1) the value of the best objective function found by HuS reached a target value of 0.05 or less; (2) the objective function value did not improve in ten consecutive iterations.

Fig. 6 shows the best and the worst history convergence for each dimension. Surprisingly, the HuS can found the optimum point for $d = 20$ in a lower number of iterations than for $d = 10$. Table 4 presents results obtained by HuS and other

Table 4
Comparative results of a high-dimensional Griewank function Eq. (7).

Algorithm	$d = 10$				$d = 20$			
	% Success	Mean S	Mean NF	Time (s)	% Success	Mean S	Mean NF	Time (s)
HuS	**100**	**0.0471**	**3166**	**0.8**	**100**	**0.0486**	**2959**	**1.3**
GAs (Evolver) [17]	50	0.06	–	312	30	0.097	–	∼1000
MAs [17]	90	0.014	–	∼40	100	0.013	–	∼100
PSO [17]	30	0.093	–	∼11	80	0.081	–	∼20
SFL [17]	50	0.08	–	∼3	70	0.063	–	∼11
Algorithm	$d = 50$				$d = 100$			
	% Success	Mean S	Mean NF	Time (s)	% Success	Mean S	Mean NF	Time (s)
HuS	**100**	**0.0480**	**5515**	**3.7**	**100**	**0.0481**	**10 029**	**11.5**
GAs (Evolver) [17]	10	0.161	–	∼2100	0	0.432	–	2427
MAs [17]	100	0.011	–	∼180	100	0.009	–	428
PSO [17]	100	0.011	–	∼40	100	0.011	–	∼300
SFL [17]	90	0.049	–	∼30	100	0.019	–	∼70

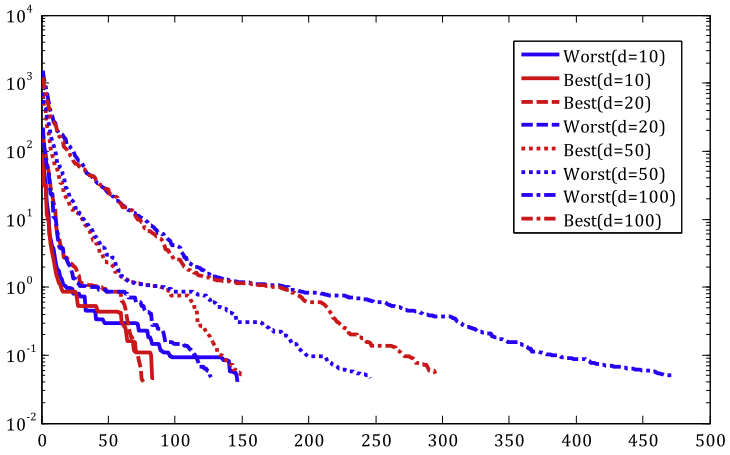

Fig. 6. Best/worst convergence history of a high-dimensional Griewank function ($d = 10, 20, 50, 100$).

compared algorithms. Reported is the percentage of success (% Success), mean solution (Mean S), mean number of function evaluations (Mean NF) and Processing time to reach the optimum (Time). Note that, for the compared algorithms, the mean number of function evaluation values are not available.
Obviously, HuS outperforms the other compared algorithms in terms of percentage of success and mean solution. The comparison of processing time may not be a fair comparison because the processor/hardware used in this study is different from that used in [17]. Unfortunately, the authors in [17] did not mention the number of function evaluations for further comparison.

3.2. Unconstrained function minimization examples—set two

The list of test functions on which we ran HuS, together with the number of dimensions used and the initialization interval, is presented in Table 5. Figures of the test functions used can be found in [18].

Following the test set-up described in the literature, we performed 100 independent runs, with the following stopping criterion (as used by the other algorithms in this comparison):

$$|f - f^*| < \epsilon_1 f + \epsilon_2, \tag{8}$$

where f is the value of the best solution found by HuS, f^* is the optimal value for the given test problem, and ϵ_1 and ϵ_2 are respectively the relative and absolute errors. For all the test simulations we have used $\epsilon_1 = \epsilon_2 = 10^{-4}$ following the values reported in the literature.

In this section, we compare our method with four other meta-heuristic optimization algorithms. These methods have some kind of cooperation between members similar to our method. They are optimization algorithms inspired by the foraging behavior of ants: ACO extended to continuous domains (ACO_R) [19], continuous ACO (CACO) [20], the API algorithm [21], and Continuous Interacting Ant Colony (CIAC) [18].

Table 5
Test functions used for comparing HuS to other mentioned algorithms Section 3.2.

Function	Formula
$B_2\ \vec{x} \in [-100, 100]^d,\ n = d$	$f_{B2}(\vec{x}) = x_1^2 + 2x_2^2 - \frac{3}{10}\cos(3\pi x_1) - \frac{2}{5}\cos(4\pi x_2) + \frac{7}{10}$
Goldstein and Price (GP) $\vec{x} \in [-2, 2]^d, d = 2$	$f_{GP}(\vec{x}) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2))$
Martin and Gaddy (MG) $\vec{x} \in [-20, 20]^d,\ d = 2$	$f_{MG}(\vec{x}) = (x_1 - x_2)^2 + \left(\frac{x_1 + x_2 - 10}{3}\right)^2$
Rosenbrock (R_n) $\vec{x} \in [-5, 10]^d,\ d = 2$	$f_{R_n}(\vec{x}) = \sum_{i=1}^{d-1} 100(x_i^2 - x_{i+1}) + (x_i - 1)^2$
Griewangk (GR_d) $\vec{x} \in [-5.12, 5.12]^d,\ d = 10$	$f_{GR_d}(\vec{x}) = \left(\frac{1}{10} + \left(\sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)\right)^{-1}$
Sphere model (SM) $\vec{x} \in [-5.12, 5.12]^d,\ d = 6$	$f_{SM}(\vec{x}) = \sum_{i=1}^{d} x_i^2$

Table 6
HuS parameter values for unconstrained problems of Section 3.2.

Parameter	Value					
	Rosenbrock (R_2)	Griewangk (GR_{10})	Goldstein and Price	Martin and Gaddy	B_2	Sphere
Maximum number of epochs (NE)	8	6	10	5	10	10
Iteration per epoch (IE)	10	25	15	20	10	10
Hunting group size (HGS)	12	10	2	2	2	2
Maximum movement toward leader (MML)	0.4	0.4	0.3	0.3	0.3	0.3
Hunting group consideration rate (HGCR)	0.2	0.3	0.3	0.3	0.3	0.3
Ra_{max}, Ra_{min}	0.25, 10^{-5}	0.005, 10^{-6}	0.25, 10^{-5}	0.2, 5×10^{-6}	10^{-2}, 5×10^{-4}	10^{-2}, 5×10^{-5}
Reorganization parameters $-\alpha$, β	0.2, −0.5	0.2, −1	0.01, −0.05	0.05, −0.05	0.005, −0.05	0.01, −0.5

Table 7
Simulation results of the unconstrained problems of Section 3.2.

Algorithm	Rosenbrock (R_2)		Griewangk (GR_{10})		Goldstein and Price	
	% Success	Mean NF	% Success	Mean NF	% Success	Mean NF
HuS	100	1 423	**84**	**1 328**	**100**	**368**
ACO_R [19]	**100**	**820**	61	1 390	100	384
CIAC [18]	100	11 480	52	50 000	56	23 420
API [21]	100	9 840	–	–	–	–
CACO [20]	100	6 800	100	50 000	100	5 370
Algorithm	Martin and Gaddy		B_2		Sphere	
	% Success	Mean NF	% Success	Mean NF	% Success	Mean NF
HuS	**100**	**315**	**100**	**327**	**100**	**323**
ACO_R [19]	100	345	100	544	100	781
CIAC [18]	20	11 730	100	11 960	100	50 000
API [21]	–	–	–	–	100	10 150
CACO [20]	100	1 725	–	–	100	21 860

Table 6 summarizes the HuS parameter values used in this problem. Table 7 presents the results obtained by HuS and other compared algorithms. Reported is the percentage of success (% Success) and mean number of function evaluations (Mean NF).

When compared with other algorithms, HuS is the winner in almost all cases in respect of percentage of success and mean number of function evaluations. In fact, HuS has proved to be more effective. Only for the Rosenbrock function does ACO_R perform better in respect of mean number of function evaluations.

3.3. Constrained function minimization examples

In this section, three constrained optimization problems from the literature are presented to show the efficiency of the HuS in handling this type of problem. Table 8 summarizes the HuS parameter values used in this section.

Table 8
HuS parameter values for constrained problems of Section 3.3.

Parameter	Value		
	Constrained function 1	Constrained function 2	Constrained function 3
Number of epochs (NE)	5	8	12
Iteration per epoch (IE)	25	100	150
Hunting group size (HGS)	10	10	10
Maximum movement toward leader (MML)	0.3	0.3	0.2
Hunting group consideration rate (HGCR)	0.3	0.3	0.3
Ra_{max}, Ra_{min}	10^{-2}, 10^{-4}	10^{-2}, 10^{-5}	10^{-2}, 10^{-4}
Reorganization parameters $-\alpha$, β	0.01, −1	0.01, −0.5	0.05, −0.05

3.3.1. Constrained function 1

$$
\begin{aligned}
&fx = (x_1 - 2)^2 + (x_2 - 1)^2 \\
&\text{subject to,} \\
&g_1(x) = x_1 - 2x_2 + 1 = 0 \\
&g_2(x) = -x_1^2/4 - x_2^2 + 1 \geq 0 \\
&-10 \leq x_1 \leq 10 \qquad -10 \leq x_2 \leq 10.
\end{aligned} \tag{9}
$$

This problem was originally introduced by Braken and McCormick [22]. The optimum solution using the HuS algorithm is obtained at $x^* = (0.82288, 0.91144)$ with an objective function value equal to $f^*(x) = 1.3935$. Homaifar et al. [23] solved the problem using GA. Fogel [24] also solved it using evolutionary programming. Lee and Geem [9] used the harmony search algorithm to find the optimum point. After 3750 searches (5 epoch and 125 movement toward the leader), the HuS algorithm found the optimum vector. Table 9 shows the best solution vector from the HuS algorithm and also the results obtained from the compared algorithms. The best vector found using the HuS meta-heuristic algorithm was $x = (0.8228\ 0.9114)$, and the corresponding objective function value was 1.3935. As can be observed, the HuS algorithm has reached a better solution compared to the other three in terms of the objective function values and the constraint accuracy.

3.3.2. Constrained function 2

$$
\begin{aligned}
&f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\
&\text{subject to,} \\
&g_1(x) = 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0 \\
&g_2(x) = 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0 \\
&g_3(x) = 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0 \\
&g_4(x) = -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0 \\
&-10 \leq x_i \leq 10 \quad (i = 1 - 7).
\end{aligned} \tag{10}
$$

This constrained problem has seven design variables, and four constraint functions. After 8 epochs (approximately 22 000 searches) the optimal solution was obtained at $x^* = (2.330875,\ 1.951370,\ -0.474593,\ 4.365553,\ -0.037936,\ 1.594065)$ with the objective function value equal to $f(x^*) = 680.6300771$. For this problem, no constraints were violated. Table 10 presents the results obtained by HuS and other compared algorithms. The solution obtained by the HuS algorithm is comparable with the previous solution reported in the literature and it take a smaller number of function evaluations to find the optimum point.

3.3.3. Constrained function 3

$$
\begin{aligned}
&f(x) = x_1 + x_2 + x_3 \\
&\text{subject to,} \\
&g_1(x) = 1 - 0.0025(x_4 + x_6) \geq 0 \\
&g_2(x) = 1 - 0.0025(x_5 + x_7 - x_4) \geq 0 \\
&g_3(x) = 1 - 0.01(x_8 - x_5) \geq 0 \\
&g_4(x) = x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \geq 0 \\
&g_5(x) = x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geq 0 \\
&g_6(x) = x_3x_8 - x_3x_5 - 2500x_5 - 11250000 \geq 0 \\
&100 \leq x_1 \leq 10000, \qquad 1000 \leq x_2 \qquad x_3 \leq 10000, \qquad 10 \leq x_i \leq 1000 \quad (i = 4 - 8).
\end{aligned} \tag{11}
$$

Table 9
Optimal results for constrained function 1 Eq. (9).

Methods	Optimal design variables (x)		Constraints		Objective function value $f(x)$
	x_1	x_2	g_1	g_2	
Homaifar et al. [23]	0.8080	0.8854	3.7×10^{-2}	5.2×10^{-2}	1.4339
Fogel [25]	0.8350	0.9125	1.0×10^{-2}	-7.0×10^{-2}	1.3772
Lee and Geem [9]	0.8343	0.9121	5.0×10^{-3}	5.4×10^{-3}	1.3770
HuS (present study)	**0.8228**	**0.9114**	$\mathbf{1.0 \times 10^{-6}}$	$\mathbf{4.9 \times 10^{-6}}$	**1.3935**

Table 10
Optimal results for constrained function 2 Eq. (10).

Optimal design variables x, $f(x^*)$ and number of searches	Michalewicz [25]	Deb [13]	Lee and Geem [9]	Fesanghary [26]	HuS (present study)
x_1	–	–	2.32345617	2.33047	**2.33087488**
x_2	–	–	1.951242	1.95137	**1.95136990**
x_3	–	–	−0.448467	−0.47772	**−0.47459349**
x_4	–	–	4.3619199	4.36574	**4.36555341**
x_5	–	–	−0.630075	−0.62448	**−0.62452549**
x_6	–	–	1.03866	1.03794	**1.03793634**
x_7	–	–	1.605384	1.59414	**1.59406525**
$f(x^*)$	680.642	680.63446	680.6413574	680.6300577	**680.6300771**
Number of searches	–	350 070	160 000	42 000	**22 000**

Table 11
Optimal results for constrained function 3 Eq. (11).

Optimal design variables x, $f(x^*)$ and number of searches	Michalewicz [25]	Deb [13]	Lee and Geem [9]	HuS (present study)
x_1	–	–	500.0038	**522.80683779**
x_2	–	–	1359.3110	**1380.64447162**
x_3	–	–	5197.9595	**5147.87099665**
x_4	–	–	174.7263	**177.10111645**
x_5	–	–	292.0817	**294.08520721**
x_6	–	–	224.7054	**222.89828045**
x_7	–	–	282.6446	**283.01588927**
x_8	–	–	392.0817	**394.08520715**
$f(x^*)$	7377.976	7060.221	7057.274414	**7051.322306**
Number of searches	N/A	320 080	150 000	**52 000**

In this problem there are eight design variables, six nonlinear constraints and 14 boundary conditions. The optimal solution is obtained at $x^* = (522.806838, 1380.644472, 5147.870997, 177.101116, 294.085207, 222.898280, 283.015889, 394.085207)$, with corresponding objective function value equal to $f(x^*) = 7051.322306$ after approximately 52 000 function evaluations (12 epochs). Table 11 compares the best solution of constrained function 3 obtained using the HuS algorithm with the previous best solution reported in the literature. It can be seen from Table 11 that the result obtained using the HuS algorithm is better than feasible solutions previously reported with respect to the objective function value and the number of function evaluations.

4. Conclusion

A new meta-heuristic algorithm has been developed. The Hunting Search (HuS) meta-heuristic optimization algorithm was conceptualized using the strategy of group hunters in catching their prey. Compared to gradient-based mathematical optimization algorithms, the HuS algorithm imposes fewer mathematical requirements and does not require initial value settings of the decision variables. In addition, the HuS algorithm uses stochastic searches; therefore, derivative information is unnecessary.

Selected benchmark optimization problems were solved to demonstrate the effectiveness and robustness of the new algorithm compared to other optimization methods. The test cases showed that the HuS algorithm is a global search algorithm that can be easily applied to various optimization problems. The results obtained using the HuS algorithm would yield better solutions than those obtained using other algorithms. In addition, in constrained optimization problems, hunters have power of orientation in the design space (movement toward leader); therefore, they can escape from forbidden areas and find feasible areas quickly, as real hunters do.

Further work is still needed to solve more complex and real optimization problems such as engineering problems. The algorithm can also be generalized for solving discrete and combinatorial optimization problems such as the traveling salesman problem and timetabling.

References

[1] L.J. Fogel, A.J. Owens, M.J. Walsh, Artificial Intelligence through Simulated Evolution, John Wiley, Chichester, UK, 1966.
[2] K. De Jong, Analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, University of Michigan, Ann Arbor, MI, 1975.
[3] J.R. Koza, Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems, Rep. No. STAN-CS-90-1314, Stanford University, CA, 1990.
[4] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[5] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, Boston, MA, 1989.
[6] F. Glover, Heuristic for integer programming using surrogate constraints, Decis. Sci. 8 (1) (1977) 156–166.
[7] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[8] Z.W. Geem, J.H. Kim, G.V. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.
[9] K.S. Lee, Z.W. Geem, A new meta-heuristic algorithm for continues engineering optimization: harmony search theory and practice, Comput. Methods Appl. Mech. Engrg. 194 (2004) 3902–3933.
[10] Zong Woo Geem, Novel derivative of harmony search algorithm for discrete design variables, Appl. Math. Comput. 199 (2008) 223–230.
[11] M. Mahdavi, M. Fesanghary, E. Damangir, An improved harmony search algorithm for solving optimization problems, Appl. Math. Comput. 188 (2007) 1567–1579.
[12] Mahamed G.H. Omran, Mehrdad Mahdavi, Global-best harmony search, Appl. Math. Comput. 198 (2008) 643–656.
[13] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Methods Appl. Mech. Engrg. 186 (2000) 311–338.
[14] R. Dawkins, The Selfish Gene, Oxford University Press, Oxford, 1976.
[15] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948.
[16] S.-Y. Liong, Md. Atiquzzaman, Optimal design of water distribution network using shuffled complex evolution, J. Inst. Eng. Singap. 44 (1) (2004) 93–107.
[17] Emad Elbeltagia, Tarek Hegazyb, Donald Griersonb, Comparison among five evolutionary-based optimization algorithms, Adv. Eng. Inf. 19 (2005) 43–53.
[18] J. Dréo, P. Siarry, Continuous interacting ant colony algorithm based on dense hierarchy, Future Gener. Comput. Syst. 20 (5) (2004) 841–856.
[19] Krzysztof Socha, Marco Dorigo, Ant colony optimization for continuous domains, European J. Oper. Res. 185 (2008) 1155–1173.
[20] B. Bilchev, I.C. Parmee, The ant colony metaphor for searching continuous design spaces, in: T.C. Fogarty (Ed.), Proceedings of the AISB Workshop on Evolutionary Computation, in: Lecture Notes in Comput. Sci., vol. 993, Springer, Berlin, 1995, pp. 25–39.
[21] N. Monmarché, G. Venturini, M. Slimane, On how pachycondyla apicalis ants suggest a new search algorithm, Future Gener. Comput. Syst. 16 (2000) 937–946.
[22] J. Bracken, G.P. McCormick, Selected Applications of Nonlinear Programming, John Wiley & Sons, New York, 1968.
[23] A. Homaifar, S.H.-V. Lai, X. Qi, Constrained optimization via genetic algorithms, Simulation 62 (4) (1994) 242–254.
[24] D.B. Fogel, A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems, Simulation 64 (6) (1995) 399–406.
[25] Z. Michalewicz, Genetic algorithms, numerical optimization, and constraints, in: L. Esheman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kauffman, San Mateo, 1995, pp. 151–158.
[26] M. Fesanghary, M. Mahdavi, M. Minary-Jolandan, Y. Alizadeh, Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems, Comput. Methods Appl. Mech. Engrg. 197 (2008) 3080–3091.