INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2012; **19**:666–684
Published online 30 May 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nme.4295

A comparison of swarm intelligence algorithms for structural engineering optimization

Rafael S. Parpinelli[1,2], Fábio R. Teodoro[1] and Heitor S. Lopes[1,*,†]

[1] *Bioinformatics Laboratory, Federal University of Technology - Paraná, Curitiba, Brazil*
[2] *Department of Computer Science, Santa Catarina State University, Joinville, Brazil*

SUMMARY

This paper compares the performance of three swarm intelligence algorithms for the optimization of hard engineering problems. The algorithms tested were bacterial foraging optimization (BFO), particle swarm optimization (PSO), and artificial bee colony (ABC). Besides the regular BFO, two other variants reported in the literature were also included in the study: adaptive BFO and swarming BFO. Both PSO and ABC were tested using the regular algorithm and variants that include explosion (mass extinction). Three optimization problems of structural engineering were used: minimization of the cost of a welded beam, minimization of the construction cost of a pressure vessel, and minimization of the total weight of a 10-bar plane truss. All problems are strongly constrained. The algorithms were evaluated using two criteria: quality of solutions and the number of function evaluations. The results show that PSO presented the best balance between these two criteria. For the optimization problems approached in this paper, we can also conclude that the explosion procedure resulted in no significant improvements. Copyright © 2012 John Wiley & Sons, Ltd.

Received 15 June 2011; Revised 4 January 2012; Accepted 14 January 2012

KEY WORDS: swarm intelligence; bacterial foraging optimization; particle swarm optimization; artificial bee colony; engineering problems

1. INTRODUCTION

Most structural engineering optimization problems are non-linear and highly constrained. Because of the limitations of exact methods in approaching such complex problems, the need for more robust techniques arises. Evolutionary computation and, more specifically, swarm intelligence (SI) provide a range of flexible and robust optimization techniques capable of dealing with this sort of problems.

In the beginning, the mainstream paradigms of the research in the area of SI were the ant colony optimization[‡] (ACO) [1] and the particle swarm optimization[§] (PSO) [2]. Later, many other algorithms and variants appeared.

The ACO metaheuristics is inspired by the foraging behavior of ants. The ants' goal is to find the shortest path between a food source and the nest. The standard ACO was conceived for dealing with problems in which the search space is discrete and can be abstracted as a graph. Consequently, each path constructed by the ants represents a potential solution to the problem being solved.

The PSO metaheuristics is motivated by the coordinate movement of fish schools and bird flocks. The PSO is compounded by a swarm of particles that interact with each other in a continuous search space. The position of each particle represents a potential solution to the problem being solved, and it is represented as an n-dimensional vector. In PSO, particles 'fly' through the hyperdimensional

*Correspondence to: Heitor S. Lopes, Bioinformatics Laboratory, Federal University of Technology - Paraná, Curitiba 80230-901, Brazil.
†E-mail: hslopes@utfpr.edu.br
‡ACO repository: http://iridia.ulb.ac.be/∼mdorigo/ACO/
§PSO Repository: http://www.particleswarm.info

search space, and changes to their positions are based on the socio-cognitive tendency of individuals to emulate the success of other individuals. Each particle of the swarm has its own life experience and is able to evaluate the quality of its experience. As social individuals, they also have knowledge about how well their neighbors have behaved. These two kinds of information correspond to the cognitive component (individual learning) and social component (cultural transmission), respectively. Hence, decisions of an individual are made by taking into account both the cognitive and the social components, thus leading the population (swarm) to an emergent behavior.

Both aforementioned methods have been applied successfully in a vast range of problems [3]. Notwithstanding, in recent years several other SI algorithms have appeared, inspired by fireflies bioluminescence, slime molds life cycle, cockroaches infestation, mosquitoes host-seeking, bats echolocation, bees mating, bees foraging, and bacterial foraging. For a detailed review about these algorithms, see [4].

In this paper, the performance of three SI algorithms (and variants of them) were tested for the optimization of hard engineering optimization problems. The empirically selected swarm-based algorithms were as follows: bacterial foraging optimization (BFO), PSO, and artificial bee colony (ABC). Besides the regular BFO, two other variants reported in the literature were also included in the study: adaptive BFO (BFO-A) and swarming BFO (BFO-S). Both PSO and ABC were tested using the regular algorithm and variants that include explosion (mass extinction).

All these approaches are global optimization metaheuristics, and they are composed, basically, of a selection of the best scheme and of a randomization scheme. The former guides the algorithm convergence to the optimality (exploitation), and the latter avoids both the loss of diversity and the algorithm to get trapped in local optima (exploration). A good balance between exploitation and exploration may lead to the global optimality achievement.

The objective of this article is not to propose a new version or modifications to algorithms but to verify the differences in exploitation and exploration balance for all algorithms performing an unbiased comparison over real problems. Three optimization problems of structural engineering were used: minimization of the cost of a welded beam, minimization of the construction cost of a pressure vessel, and minimization of the total weight of a 10-bar plane truss. As mentioned before, these problems are strongly constrained.

This paper is organized as follows: the next section describes all the SI algorithms used in this work. Section 3 describes the experiments and the optimization problems. Section 4 shows the results of the comparisons between algorithms. Finally, Section 5 presents the conclusions and points future research directions.

2. SWARM INTELLIGENCE ALGORITHMS

Swarm-based algorithms are inspired by the behavior of some social living beings, such as ants, bees, birds, and fishes. Self-organization and decentralized control are remarkable features of swarm-based systems that, such as in nature, leads to an emergent behavior. Emergent behavior is a property that emerges through local interactions among system components, and it is not possible to be achieved by any of the components of the system acting alone [5, 6].

In this work, we compared the performance of three SI algorithms and some variants. These algorithms are presented in detail in the next sections.

2.1. Bacterial foraging optimization

In this section, the canonical BFO algorithm is described. Later, some proposed variants of this algorithm are presented: the BFO-A and the BFO-S. The BFO algorithm has been applied to several optimization problems, such as multi-objective optimization [7]. See [8] for a summary of recent applications.

The bacterial foraging algorithm was first proposed by [9], and it is inspired by the foraging behavior of *Escherichia coli* bacteria. Possible solutions to an optimization problem are represented in the BFO algorithm by a colony of S bacteria of dimension d, set in the search space.

Algorithm 1 Canonical BFO

```
1: Initialize control parameters: $S, Nc, Ns, Nre, Ned, Ped, Sr, C^i, \theta^i (i = 1, 2, \cdots, S)$ {$\theta^i$ represents the position of the $i$-th bacterium}
2: $l \leftarrow 0$
3: repeat
4:   $k \leftarrow 0$
5:   repeat
6:     $j \leftarrow 0$
7:     repeat {Chemotaxis phase}
8:       for all $i$ do
9:         $J_{last} \leftarrow J(i, j, k, l)$ {fitness of $i$-th bacterium}
10:        Generate a random unity vector: $\Delta^i$
11:        $\theta^i(j+1, k, l) \leftarrow \theta^i(j, k, l) + \Delta^i C^i$ {new position of the bacterium}
12:        Compute $J(i, j+1, k, l)$ with $\theta^i(j+1, k, l)$ {new fitness fitness of the bacterium}
13:        $m \leftarrow 0$
14:        while $m < Ns$ and $J(i, j+1, k, l) < J_{last}$ do {swim}
15:          $m \leftarrow m + 1$
16:          Move the bacterium one more step, of length $C^i$, in the direction $\Delta^i$
17:          Compute $J(i, j+1, k, l)$ with the new $\theta^i(j+1, k, l)$
18:        end while
19:      end for
20:      $j \leftarrow j + 1$
21:    until $j \geq Nc$
22:    for all $i$ do {Computes the "health" of bacteria}
23:      $J^i_{health} \leftarrow \sum_{j=0}^{Nc} J(i, j, k, l)$
24:    end for
25:    Substitute the $Sr$ bacteria with highest $J_{health}$ by copies of the $Sr$ bacteria with the smallest $J_{health}$ {reproduction phase}
26:    $k \leftarrow k + 1$
27:  until $k \geq Nre$
28:  Set a random position in the search space for each bacteria, with probability $Ped$ {dispersion phase}
29:  $l \leftarrow l + 1$
30: until $l \geq Ned$
```

Algorithm 1 shows the basic BFO, as described by [10]. It consists of three main phases, explained later: chemotaxis, reproduction, and elimination–dispersal.

In the chemotaxis phase, every bacterium moves a single step towards a random position. If an improvement in the fitness (decrement) of the bacterium comes out from this movement, a swim is done. This means that the bacterium keeps on moving in the same direction while the fitness is decreasing or until a maximum number of swim steps (N_s) is reached. This is done for all bacteria. In this version of the algorithm, the step is set to $C(i)$.

In the reproduction phase, the healthiest S_r bacteria are replicated and introduced in the same region of the search space where their 'parents' were from. Conversely, the S_r least healthy bacteria are eliminated. The health of a bacterium is the summation of the fitness for all chemotaxis steps just done. The smaller the value, the healthier the bacteria and, consequently, the better the solution it represents.

In the dispersion phase, each bacterium can be eliminated and substituted by another one, randomly generated, according to a given probability P_{ed}.

By default, BFO is a minimization algorithm and has the following control parameters:

- S, number of bacteria;
- N_c, number of chemotaxis steps;
- N_s, maximum number of swim steps;
- N_{re}, number of reproduction steps;
- N_{ed}, number of dispersion steps;
- P_{ed}, probability of dispersion;
- $C(i)(i = 1, 2, \ldots, S)$, step size of the bacteria;
- S_r, number of bacteria replicated at each reproduction step.

2.1.1. Adaptive bacterial foraging optimization. Chen *et al*. [10] observed that the quality of solutions provided by the BFO algorithm is strongly dependent of the step size of the bacteria. Therefore, they proposed a variant, known as adaptive BFO, to circumvent this problem. In this variant, as the step size is individualized for each bacterium, the overall behavior of each bacterium can be significantly different from each other. The behavior can alternate between global search (when the step size is larger) when it is stagnated in a limited region of the search space and local search (when the step size is smaller) when it is in a promising region.

Adaptive BFO includes some other parameters to the basic BFO: α, β, ε, and N_u. Just after the chemotaxis step of each bacterium, a stagnation counter (*no_improv_count*^{i}) is updated: it is incremented if the bacterium does not improve its fitness or zeroed otherwise. The parameter α is the reduction factor of the step size (C^i). The parameter β is the reduction factor of another parameter ε, which, in turn, is the threshold (of the fitness value) from which the bacterium has its step size reduced. The parameter N_u is the number of chemotaxis steps that a bacterium is allowed to stagnate before returning to the global search mode. If the fitness of a bacterium is improved after the chemotaxis step at time t, its size is updated for the next iteration ($C^i(t+1)$) according to Equation (1). The parameter ε^i associated to this bacterium is updated according to Equation (2).

$$C^i(t+1) = \begin{cases} C^i(t)/\alpha & \text{, if } J(i,j,k,l) < \varepsilon^i(t) \\ C^i(t) & \text{, if } J(i,j,k,l) \geqslant \varepsilon^i(t) \end{cases} \tag{1}$$

$$\varepsilon^i(t+1) = \begin{cases} \varepsilon^i(t)/\beta & \text{, if } J(i,j,k,l) < \varepsilon^i(t) \\ \varepsilon^i(t) & \text{, if } J(i,j,k,l) \geqslant \varepsilon^i(t) \end{cases} \tag{2}$$

If no improvement of the fitness occurs after the chemotaxis step, parameters C^i and ε^i are updated according to Equations (3) and (4). Eventually, a bacterium can return to the global search behavior.

$$C^i(t+1) = \begin{cases} C^i_{\text{initial}} & \text{, if } \textit{no\_improv\_count}^i > N_u \\ C^i(t) & \text{, if } \textit{no\_improv\_count}^i \leqslant N_u \end{cases} \tag{3}$$

$$\varepsilon^i(t+1) = \begin{cases} \varepsilon^i_{\text{initial}} & \text{, if } \textit{no\_improv\_count}^i > N_u \\ \varepsilon^i(t) & \text{, if } \textit{no\_improv\_count}^i \leqslant N_u \end{cases} \tag{4}$$

2.1.2. Swarming bacterial foraging optimization. Some species of bacteria, such as *E. coli*, have a group behavior when moving together. This phenomenon is known as swarming, and it is a consequence of pheromones that act as attractants for the bacteria. On the basis of this fact, [9] proposed a variant of BFO named swarming BFO, in which a bacterium attracts the other ones by modifying the search space around itself according to Equation (5). This behavior reflects the tendency of the real bacteria to stay close to each other. They also repel other bacteria that are too close, as an analogy to the nutrient consumption by bacteria, keeping a minimum distance between them.

Four new control parameters are introduced in BFO-S: d_{attract} represents the depth of the attractor released by the bacteria; w_{attract} corresponds to the width of the attractant signal; $h_{\text{repellant}}$ and $w_{\text{repellant}}$ are, respectively, the height and width of the repellant.

The distortion of the search space where the bacteria move is accomplished with Equation (5), added to the fitness function. Its effect makes bacteria stay close to each other.

$$\begin{aligned} J_{cc}(\theta, P(j,k,l)) &= \sum_{i=1}^{S} J_{cc}^{i}(\theta, \theta^{i}(j,k,l)) \\ &= \sum_{i=1}^{S} \left[-d_{\text{attract}} \cdot e^{\left(-w_{\text{attract}} \sum_{m=1}^{p} (\theta_m - \theta_m^i)^2 \right)} \right] \\ &\quad + \sum_{i=1}^{S} \left[h_{\text{repellant}} \cdot e^{\left(-w_{\text{repellant}} \sum_{m=1}^{p} (\theta_m - \theta_m^i)^2 \right)} \right] \end{aligned} \tag{5}$$

In this equation, θ represents a position in the search space; $P(j,k,l)$ is the set of positions of all bacteria in the jth chemotaxis step of the kth reproduction step of the lth dispersion step; θ^i is the position of bacterium i; θ_m represents component m of the vector concerning position θ; and θ_m^i is the component m of θ^i.

2.2. *Particle swarm optimization*

The PSO metaheuristic was inspired by the coordinate movement of fish schools and bird flocks [2] and has been applied to several optimization problems (for instance, [11–13]).

The PSO is a population-based metaheuristic composed of a swarm of n particles. Each particle represents a potential solution to the problem to be solved. The position of a particle in the search space is determined by the solution it currently represents. Algorithm 2 shows the canonical PSO, as described by [2]. In this algorithm, each solution $\vec{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ of dimension d is evaluated by a fitness function $f(\vec{x}_i)$, $i = 1, \ldots, n$. In PSO, particles 'fly' through the hyperdimensional search space according to their velocity $\vec{v}_i$. Changes to the position of the particles within the search space are based on the socio-cognitive tendency of individuals to emulate the success of other individuals. Each individual of a population has its own life experience ($\vec{p}_i$) and is able to evaluate the quality of its experience. They are social individuals, and so they also have knowledge about the quality of their neighbors ($\vec{g}$). These two sources of information correspond to the cognitive component (individual learning) and social component (social learning), respectively. Hence, an individual decision is made considering both the cognitive and the social components, thus leading the population to an emergent behavior of navigating coordinately through the search space.

The parameters φ_p and φ_g determine the relative influence of the cognitive and social components, respectively, and are often both set to the same value so as to give each component (the cognition and social learning rates) the same decisional weight. The stochastic nature of PSO is evidenced by r_p and r_g that are numbers randomly generated each time the equation is computed.

PSO, like other evolutionary computation algorithms, may converge to local optima during the search. Sometimes this can be avoided or, at least, delayed, adapting accordingly the control parameters. However, after being converged, the chance of finding better solutions is very poor. This is a consequence of a population of particles with very low diversity. Whenever the stagnation of the search is detected, some corrective measure should be taken; otherwise few or no improvements can be expected. This can be done through a mechanism known as mass extinction, decimation and hot-boot, or, simply, explosion. This method has already been used in PSO [11], genetic algorithms [14], and other Evolutionary Computation algorithms (for instance, [15]), and it has been demonstrated to improve the performance of the basic algorithm.

Algorithm 2 Canonical PSO

```
1: Set parameters: $n,\ \varphi_p,\ \varphi_g$
2: for $i = 1$ to $n$ do
3:     Initialize the positions $\vec{x}_i$ and velocities $\vec{v}_i$ randomly
4:     Evaluate fitness $f(\vec{x}_i)$
5:     Initialize the particle's best known position to its initial position: $\vec{p}_i = \vec{x}_i$
6:     if $f(\vec{p}_i)$ is better than $f(\vec{g})$ then
7:         Update the swarm's best known position: $\vec{g} = \vec{p}_i$
8:     end if
9: end for
10: while stop condition not met do
11:     for $i = 1$ to $n$ do
12:         Update particles' velocity: $\vec{v}_i = \vec{v}_i + \varphi_p * r_p * (\vec{p}_i - \vec{x}_i) + \varphi_g * r_g * (\vec{g} - \vec{x}_i)$
13:         Update particles' position: $\vec{x}_i = \vec{x}_i + \vec{v}_i$
14:         if $f(\vec{x}_i)$ is better than $f(\vec{p}_i)$ then
15:             $\vec{p}_i = \vec{x}_i$
16:             if $f(\vec{p}_i)$ is better than $f(\vec{g})$ then
17:                 $\vec{g} = \vec{p}_i$
18:             end if
19:         end if
20:     end for
21: end while
22: Postprocess results and visualization
```

Therefore, a variant of PSO, named PSO with explosion (PSO-X), was implemented and tested. Basically, when stagnation of the best solution is observed for a predefined number of iterations (it_x), all, but the best (p_i), particles are extinguished, and the search is re-initialized. We established the stagnation threshold of 10% of the total number of iterations. Overall, the explosion procedure allows the algorithm to be more effective so that the best solution can be searched gradually during the iterations, avoiding getting trapped in local optima.

2.3. *Artificial bee colony algorithm*

The ABC algorithm was inspired in the foraging behavior of honey bees. ABC was first proposed by [16] for solving multi-dimensional and multi-modal optimization problems. A recent work [17] compared the ABC algorithm performance against other population-based algorithms (genetic algorithm, PSO, differential evolution, and evolution strategies) upon several benchmark functions. Results showed that the performance of the ABC was better than or similar to those of the other algorithms. Another relevant work concerning the ABC algorithm analyzed the tuning of control parameters [18].

The ABC algorithm begins with n solutions (food sources) of dimension d that are modified by the artificial bees. In the same way as other evolutionary algorithms, each solution $\vec{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ is evaluated by a fitness function $f(\vec{x}_i)$, $i = 1, \ldots, n$. The bees aim at discovering places of food sources (that is, regions in the search space) with high amount of nectar (good fitness values, meaning good solutions for the problem). There are three types of bees: scout bees that randomly fly in the search space without guidance, employed bees that exploit the neighborhood of their locations selecting a random solution to be perturbed, and onlooker bees that use the population fitness to select probabilistically a guiding solution to exploit its neighborhood. If the amount of nectar of a new source is higher than that of the previous one in their memory, they update the new position and forget the previous one (this is a greedy selection method). If a solution is not improved by a predetermined number of trials, controlled by the parameter *limit*, then the food source is abandoned by the corresponding employed bee, and it becomes a scout bee.

Algorithm 3 Canonical ABC

```
1: Set parameters: $n,\ limit$
2: Initialize the food sources $\vec{x}_i$ randomly
3: Evaluate fitness $f(\vec{x}_i)$ of the population
4: $count_i = 0$
5: while stop condition not met do
6:   for $i = 1$ to $n/2$ do $\{Employed\, phase\}$
7:     Select $k, j$ and $r$ at random such that $k \in \{1, 2, ..., n\}, j \in \{1, 2, ..., d\}$,
8:     $r \in [0, 1]$
9:     $\vec{v} = x_{ij} + r \cdot (x_{ij} - x_{kj})$
10:    Evaluate solutions $\vec{v}$ and $\vec{x}_i$
11:    if $f(\vec{v})$ is better than $f(\vec{x})$ then
12:      Greedy selection
13:    else
14:      $count_i = count_i + 1$
15:    end if
16:  end for
17:  for $i = n/2 + 1$ to $n$ do $\{Onlooker\, phase\}$
18:    Calculate selection probability
19:      $P(\vec{x}_k) = \frac{f(\vec{x}_k)}{\sum_{k=i}^{n} f(\vec{x}_k)}$
20:    Select a bee using the selection probability
21:    Produce a new solution $\vec{v}$ from the selected bee
22:    Evaluate solutions $\vec{v}$ and $\vec{x}_i$
23:    if $f(\vec{v})$ is better than $f(\vec{x})$ then
24:      Greedy selection
25:    else
26:      $count_i = count_i + 1$
27:    end if
28:  end for
29:  for $i = 1$ to $n$ do $\{Scout\, phase\}$
30:    if $count_i > limit$ then
31:      $\vec{x}_i =$ random
32:      $count_i = 0$
33:    end if
34:  end for
35:  Memorize the best solution achieved so far
36: end while
37: Postprocess results and visualization
```

The ABC algorithm attempts to balance exploration and exploitation using the employed and onlooker bees to perform local search and the scout bees to perform global search, respectively. The canonical ABC is shown in Algorithm 3, and further information about the ABC algorithm can be found in the repository[¶].

Some applications found in the literature using the ABC algorithm include the generalized assignment problem optimization [19], energy distribution network configuration [20], neural network training [21], data clustering [22], solving integer programming benchmarks [23], template matching in digital images [24], clustering [25], and signal model parameter extraction [26].

Similar to the PSO, a variant of the original algorithm including an explosion (mass extinction) procedure was also implemented. This simple procedure has been shown more effective than the original ABC [24]. As in the PSO-X, in the ABC with explosion (ABC-X), when stagnation of the

[¶]ABC Repository: http://mf.erciyes.edu.tr/abc/

best solution for a certain number of iterations (it_x) is observed, the explosion takes place. Explosion maintains the best results obtained so far and extinguishes the remaining population and then restarts the search. We established the stagnation factor of about 10% of the total number of iterations. In the same way as before, the explosion procedure is intended to avoid getting the algorithm trapped in local optima.

3. COMPUTATIONAL EXPERIMENTS

In this section, we report the experiments done for comparing the several aforementioned algorithms: BFO, BFO-A, BFO-S, ABC, ABC-X, PSO, and PSO-X. We selected as test problems three hard optimization problems from the area of structural engineering: minimization of the total cost of a welded beam, minimization of the construction cost of a pressure vessel, and minimization of the weight of a 10-bar planar truss. Because of its characteristics, constraints, and complexity, these problems have been frequently used in the literature for testing metaheuristics.

All problems approached in this work are highly constrained, as most interesting engineering problems also are. To handle constraints, all algorithms use a penalty strategy commonly used in evolutionary computation [27]. If a candidate solution violates a constraint, the cost function value is penalized proportionally by decreasing its chances to be chosen. Therefore, the generalized fitness function ($\mathrm{fit}(\vec{x})$) of a given candidate solution ($\vec{x}$) is defined by Equation (6):

$$\mathrm{fit}(\vec{x}) = \overline{f(\vec{x})} + r \times \sum_{i=1}^{m} \Phi[\overline{g_i(\vec{x})}] \tag{6}$$

where $\overline{f(\vec{x})}$ represents the normalized cost function; r indicates the penalty coefficient, empirically adjusted for each problem; m is the number of constraints; $\overline{g_i(\vec{x})}$ is the normalized value for constraint i; and Φ represents the penalty function, as suggested by [27].

All experiments were done using the same type of desktop computers with core-2 Quad processor running at 2.8 GHz, 2 GB of RAM, under a minimal installation of Arch Linux. All algorithms were implemented in ANSI-C programming language. The statistical analysis was done using the R statistical computing environment.

3.1. Welded beam

In this problem, the objective is to minimize the cost of construction of a welded beam, which includes the setup, welding labor, and material costs (Figure 1). The total cost is defined by Equation (7), where $h = x_1$ and $l = x_2$ represent the thickness and the length of the welded beam, respectively, and $t = x_3$ and $b = x_4$ are the height and the width of the welded cantilever, respectively [28]. All variables represent lengths in inches.

$$f(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2) \tag{7}$$

Subject to the following constraints:

$$g_1(\vec{x}) = 13,600 - \tau(\vec{x}) \geqslant 0, \tag{8}$$

$$g_2(\vec{x}) = 30,000 - \sigma(\vec{x}) \geqslant, \tag{9}$$

$$g_3(\vec{x}) = x_4 - x_1 \geqslant 0, \tag{10}$$

$$g_4(\vec{x}) = Pc(\vec{x}) - 6000 \geqslant 0, \tag{11}$$

$$0.125 \leqslant x_1 \leqslant 10, \tag{12}$$

$$0.1 \leqslant x_2 \leqslant 10, \tag{13}$$

$$0.1 \leqslant x_3 \leqslant 10, \tag{14}$$

$$0.1 \leqslant x_4 \leqslant 5. \tag{15}$$

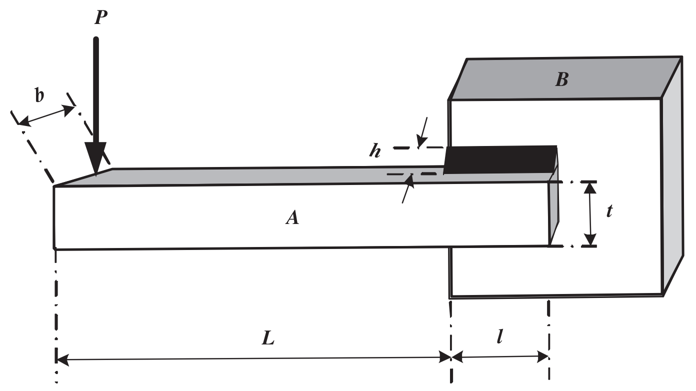

Figure 1. Welded beam.

where: $\tau(\vec{x})$, $\sigma(\vec{x})$, and $Pc(\vec{x})$ represent, respectively, the shear stress, the bending stress (measured in psi), and the buckling load (measured in pounds). The functions are defined by the following:

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + (\tau'')^2 + \frac{(x_2\tau'\tau'')}{\tau'''}}, \tag{16}$$

$$\tau' = \frac{6000}{\sqrt{2}x_1x_2}, \tag{17}$$

$$\tau'' = \frac{6000(14 + 0.5x_2)\tau'''}{2(0.707x_1x_2(x_2^2/12 + 0.25(x_1 + x_3)^2))}, \tag{18}$$

$$\tau''' = \sqrt{0.25(x_2^2 + (x_1 + x_3)^2)}, \tag{19}$$

$$\sigma(\vec{x}) = \frac{504,000}{x_3^2x_4}, \tag{20}$$

$$P_c(\vec{x}) = 64,746.022(1 - 0.0282346x_3)x_3x_4^3. \tag{21}$$

Many different mathematical optimization algorithms were proposed for solving this problem [29], as well as metaheuristics such as genetic algorithms [30] and harmony search [31]. To date, the best-known result for this problem is $f(\vec{x}) = 1.7248$, obtained by both [32] and [33]. All algorithms used a penalty coefficient $r = -1$.

3.2. Pressure vessel

This problem is the design of the cylindrical pressure vessel shown in Figure 2. The total cost that comprises not only materials but also forming and welding is to be minimized by adjusting four parameters: $T_s = x_1$ and $T_h = x_2$, the thickness of the shell and the head, respectively; $R = x_3$, the inner radius; and $L = x_4$, the length of the cylindrical section. Both L and R are continuous, but the other variables are discretized in 0.0625-in. steps.

The cost function is defined by Equation (22).

$$\begin{aligned} f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_1x_3^2 \\ +3.1661x_1^2x_4 + 19.84x_1^2x_3 \end{aligned} \tag{22}$$

Subject to the following constraints:

$$g_1(\vec{x}) = -x_1 + 0.0193x_3 \leqslant 0, \tag{23}$$

$$g_2(\vec{x}) = -x_2 + 0.00954x_3 \leqslant 0, \tag{24}$$

$$g_3(\vec{x}) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1,296,000 \leqslant 0, \tag{25}$$

$$g_4(\vec{x}) = x_4 - 240 \leqslant 0. \tag{26}$$

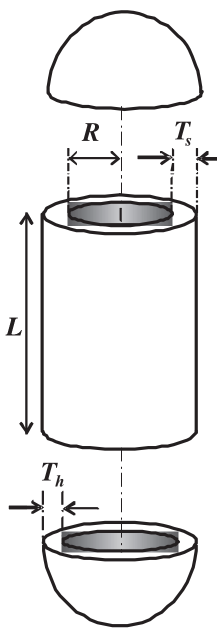

Figure 2. Pressure vessel.

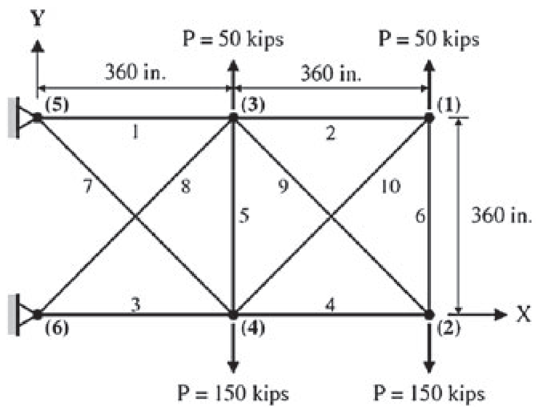

Figure 3. Ten-bar plane truss.

This problem was also solved by [31, 33–37]. The best-known result for this problem is $f(\vec{x}) = 5849.76169$, obtained by [33]. All algorithms used a penalty coefficient $r = -20$.

3.3. *Ten-bar plane truss*

In this problem, the objective is to minimize the weight of a plane truss composed of 10 bars (Figure 3). The weight can be computed by function 27:

$$f(\vec{x}) = \sum_{j=1}^{10} \rho A_j L_j \tag{27}$$

where A_j is the cross section area of the jth bar, L_j is the length of this bar, and ρ is the density of the material.

This problem was also extensively used in the optimization literature as a benchmark, mainly using conventional mathematical methods (for instance, [38–41]) or metaheuristics [31, 32]. In this work, we used the same constants and constraints used in the aforementioned works, and the best solution known is $f(\vec{x}) = 4668.72$, obtained by [32]. All algorithms used a penalty coefficient $r = -10,000$.

3.4. *Control parameters*

Because of the stochastic nature of the algorithms compared in this work, for each algorithm mentioned in Section 2, 30 independent runs were done with different initial random seeds. For each run, an upper bound for the number of fitness function evaluations was established to 500, 000.

3.4.1. Bacterial foraging optimization. Table I shows the parameters used in the BFO algorithm, which are, respectively, the population size (S), the number of chemotactic steps (N_c), the maximum number of swim steps (N_s), the number of reproduction steps (N_{re}) , the number of dispersion steps (N_{ed}), the probability of dispersion (P_{ed}), the length of the bacteria steps (C), and the proportion of the population that is replicated in the reproduction step (S_r). These parameters are the same used in [9].

The value -1 for N_{ed} means that it will continue until the stopping criterion is met, which is represented here by 500, 000 function evaluations.

3.4.2. Adaptive bacterial foraging optimization. The BFO-A uses the same parameters as BFO plus the additional parameters used by [10]. These extra parameters are, respectively (Table II), the initial threshold to enter in exploitation mode, the reduction factor to be applied on the bacteria step length (α), the reduction factor (β) applied to the threshold (ε) when this threshold is exceeded, and the number of chemotactic steps without improvement to return the bacterium to its exploration mode (N_u).

The parameters used for BFO-A experiments are shown in Table II.

Table I. Parameters used for BFO.

S	=	100
N_c	=	100
N_s	=	4
N_{re}	=	5
N_{ed}	=	−1
P_{ed}	=	0.25
C	=	0.1
S_r	=	0.5

The values recommended by [9].

Table II. Parameters used for BFO-A.

S	=	100
N_c	=	100
N_s	=	4
N_{re}	=	5
N_{ed}	=	−1
P_{ed}	=	0.25
C	=	0.1
S_r	=	0.5
ε	=	100
α	=	10
β	=	10
N_u	=	20

The values recommended by [10].

3.4.3. Swarming bacterial foraging optimization. Table III shows the parameters used for BFO-S, which are the same used for BFO plus the extra parameters of BFO-S. The extra parameters used by BFO-S are the depth of the attractant (d_{attract}), the width of the attractant (w_{attract}), the height of the repellant ($h_{\text{repellant}}$), and the width of the repellant ($w_{\text{repellant}}$).

Although w_{attract} has a smaller value than the value of $w_{\text{repellant}}$, looking at Equation (5), we could see that smaller values of w_{attract} cause a wider range of influence in the search space.

3.4.4. Particle swarm optimization. Table IV shows the parameters used by the PSO algorithm; that is, the population size (n), the relative influence of the cognitive component (φ_p), and the relative influence of the social component (φ_g).

3.4.5. Particle swarm optimization with explosion. The PSO with explosion uses the same parameters of PSO plus the number of iterations without improvement before it does the explosion (it_x). In the explosion, it re-initializes all particles positions keeping only the global best position. The parameters used for PSO-X can be seen in Table V.

3.4.6. Artificial bee colony. ABC uses as parameters the population size (n), the number of food sources (*FoodNumber*), and the number of iterations without improvement before it replaces a scout bee ($limit$) (Table VI).

Table III. Parameters used for BFO-S.

S	=	100
N_c	=	100
N_s	=	4
N_{re}	=	5
N_{ed}	=	−1
P_{ed}	=	0.25
C	=	0.1
S_r	=	0.5
d_{attract}	=	0.1
w_{attract}	=	0.2
$h_{\text{repellant}}$	=	0.1
$w_{\text{repellant}}$	=	10

The values recommended by [9].

Table IV. Parameters used for PSO.

n	=	50
φ_p and φ_g	=	2.05

Table V. Parameters used for PSO-X.

n	=	50
φ_p and φ_g	=	2.05
it_x	=	1000

Table VI. Parameters used for ABC.

n	=	50
FoodNumber	=	25
limit	=	100

3.4.7. Artificial bee colony with explosion. The parameters for ABC-X, as can be seen in Table VII, are the same as those for ABC plus the minimum number of iterations to do explosion (it_x).

4. RESULTS AND DISCUSSION

In this section, we present results of our experiments and a comparison of performance among all approaches. The performance of each approach takes into account the average best solution found in each run and the average processing time. Also, some statistical tests are performed.

4.1. Welded beam design problem

Table VIII shows the results for the welded beam problem. The PSO algorithm obtained the best mean result, which is approximately 0.13% distant of the best result known.

The convergence of the algorithms for the welded beam problem can be seen in Figure 4. In this figure, we can notice that all algorithms converge fast, particularly the PSO and PSO-X algorithms.

Figure 5 shows the Pareto's graphic with results grouped by algorithm. The PSO algorithm is the group with the best balance between mean number of function evaluations and the mean value of *fitness*. BFO is the group with the smaller number of function evaluations to converge. All groups of algorithms have its variants close to each other, and each group is distant to the others. This reinforces the fact that the employed algorithms use different search strategies.

4.2. Pressure vessel design problem

In Table IX, we can see that PSO is the algorithm that first converges and has better results. The results show that the PSO algorithm obtained the best mean result, which is approximately 3.69% distant of the best result known.

Table VII. Parameters used for ABC-X.

n	=	50
FoodNumber	=	25
limit	=	100
it_x	=	1000

Table VIII. Results for the welded beam design problem.

BFO	$fitness_{avg}$	1.92 ± 0.0
	$evaluations_{avg}$	$140,137.6 \pm 159,962.2$
BFO-A	$fitness_{avg}$	1.93 ± 0.0
	$evaluations_{avg}$	$124,448.4 \pm 142,905.1$
BFO-S	$fitness_{avg}$	1.94 ± 0.0
	$evaluations_{avg}$	$143,683.8 \pm 157,618.7$
PSO	$fitness_{avg}$	1.72 ± 0.0
	$evaluations_{avg}$	$138,215.5 \pm 78,825.1$
PSO-X	$fitness_{avg}$	1.73 ± 0.0
	$evaluations_{avg}$	$217,863.0 \pm 123,851.3$
ABC	$fitness_{avg}$	1.92 ± 0.0
	$evaluations_{avg}$	$250,697.9 \pm 121,670.6$
ABC-X	$fitness_{avg}$	1.95 ± 0.0
	$evaluations_{avg}$	$323,885.9 \pm 147,617.6$

Showing the mean *fitness* and the mean number of *evaluations* to find the best solution.

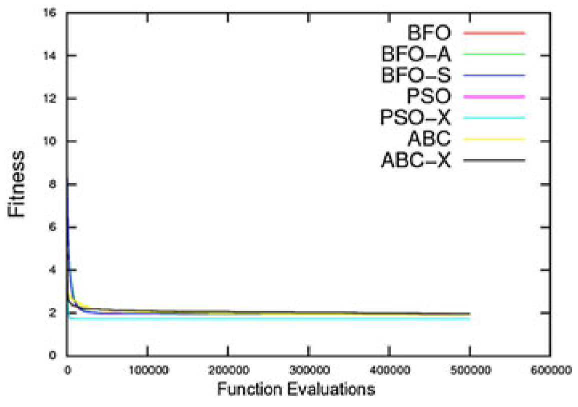

Figure 4. Convergence graphics for the welded beam design problem.

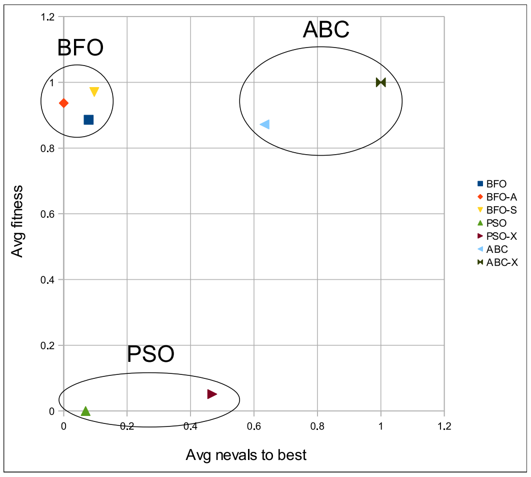

Figure 5. Pareto for the welded beam design problem.

Table IX. Results for the pressure vessel design problem.

BFO	*fitness*$_{\text{avg}}$	$11,395.8 \pm 2525.5$
	evaluations$_{\text{avg}}$	$200,786.8 \pm 133,382.1$
BFO-A	*fitness*$_{\text{avg}}$	$11,174.6 \pm 2350.7$
	evaluations$_{\text{avg}}$	$229,979.1 \pm 150,722.4$
BFO-S	*fitness*$_{\text{avg}}$	$11,587.3 \pm 2770.1$
	evaluations$_{\text{avg}}$	$228,419.3 \pm 158,479.0$
PSO	*fitness*$_{\text{avg}}$	6066.2 ± 32.0
	evaluations$_{\text{avg}}$	$75,782.0 \pm 79,598.0$
PSO-X	*fitness*$_{\text{avg}}$	6066.7 ± 32.1
	evaluations$_{\text{avg}}$	$106,104.5 \pm 114,145.9$
ABC	*fitness*$_{\text{avg}}$	6084.0 ± 15.1
	evaluations$_{\text{avg}}$	$175,303.7 \pm 91,018.9$
ABC-X	*fitness*$_{\text{avg}}$	6092.6 ± 11.9
	evaluations$_{\text{avg}}$	$329,389.9 \pm 109,327.5$

Showing the mean *fitness* and the mean number of *evaluations* to find the best solution.

Figure 6 shows the convergence of all approaches. We can see that BFO and its variants have a smoother convergence but have very poor results in comparison with the other algorithms.

Figure 7 shows that PSO variants presented the best balance between number of function evaluations and the quality of the results. The BFO variants are dominated by the others.

4.3. Ten-bar plane truss problem

In Table X, we can see that PSO-X has the best mean result, which is approximately 0.17% distant from the best result known. It also can be seen that ABC is the algorithm that first converges, but it achieves results worse than PSO-X.

All algorithms showed similar convergence curves for the 10-bar plane truss problem, with exception of the BFO algorithms which have a smoother curve, but at the end all achieved very similar results (Figure 8).

Figure 9 shows the Pareto's graphic for the 10-bar plane truss problem. ABC group is the group with the smaller number of function evaluations, and PSO group has the best results. The BFO group is dominated by the ABC group.

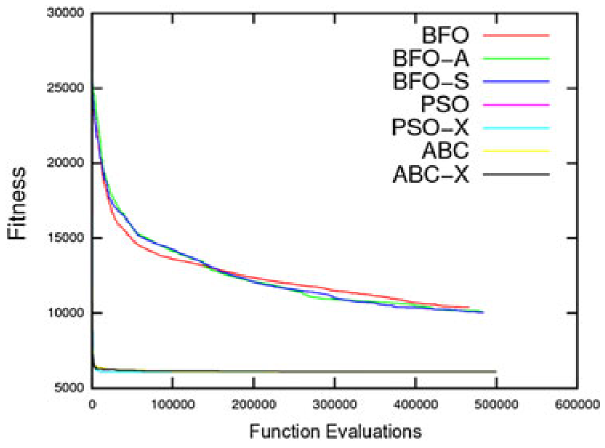

Figure 6. Convergence graphics for the pressure vessel design problem.

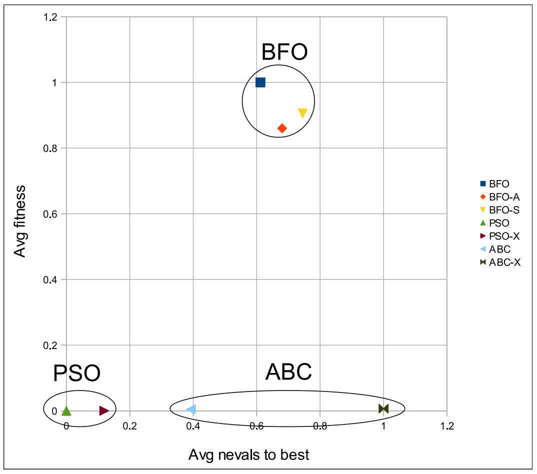

Figure 7. Pareto for the pressure vessel design problem.

Table X. Results for the 10-bar plane truss problem.

BFO	*fitness*$_{avg}$	4718.8 ± 19.1
	evaluations$_{avg}$	$449,633.9 \pm 40,472.6$
BFO-A	*fitness*$_{avg}$	4720.6 ± 15.7
	evaluations$_{avg}$	$446,626.2 \pm 47,190.9$
BFO-S	*fitness*$_{avg}$	4720.7 ± 25.4
	evaluations$_{avg}$	$446,497.7 \pm 40,385.9$
PSO	*fitness*$_{avg}$	4677.0 ± 0.0
	evaluations$_{avg}$	$429,266.0 \pm 67,654.2$
PSO-X	*fitness*$_{avg}$	4677.0 ± 0.0
	evaluations$_{avg}$	$476,990.0 \pm 19,685.2$
ABC	*fitness*$_{avg}$	4703.4 ± 9.8
	evaluations$_{avg}$	$235,727.4 \pm 140,919.5$
ABC-X	*fitness*$_{avg}$	4702.6 ± 10.7
	evaluations$_{avg}$	$250,188.1 \pm 148,331.7$

Showing the mean *fitness* and the mean number of *evaluations* to find the best solution.

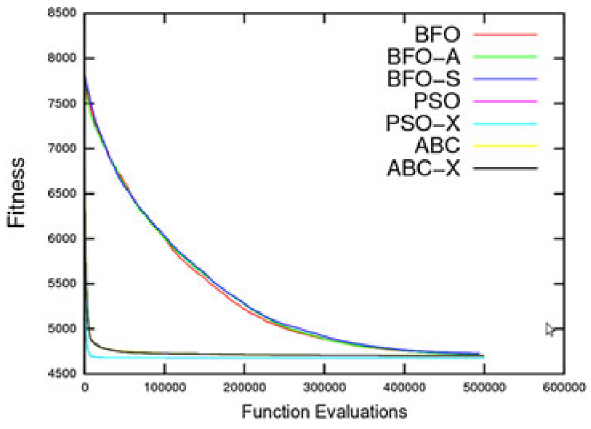

Figure 8. Convergence graphics for the 10-bar plane truss problem.

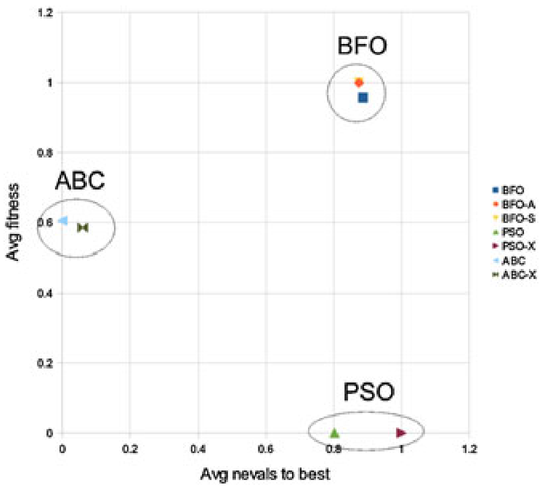

Figure 9. Pareto for the 10-bar plane truss problem.

4.4. Statistical tests

In order to properly analyze the results reported before, we performed the analysis of variance (ANOVA) test [42] upon the mean fitness data. As a prerequisite to the ANOVA test, the Shapiro–Wilk normality test [42] was run to check if the samples fall into a normal distribution. As a result, the Shapiro–Wilk test accepted the normality hypothesis with p-values equal to 2.2×10^{-16}, 2.2×10^{-16}, and 3.7×10^{-9} for welded beam design, pressure vessel design, and 10-bar plane truss problems, respectively.

The results obtained with the ANOVA test were p-values equal to 0.1246, 2.1×10^{-16}, and 1.4×10^{-8} for the three problems, respectively. Concerning the level of significance of 10%, it is possible to conclude that the null hypothesis is true (i.e., there is no difference between algorithms) only for the welded beam design problem. Table VIII shows that this statistical inference is correct.

However, for the other two problems (pressure vessel design and 10-bar plane truss), the null hypothesis is false, meaning that there are significant differences between algorithms. This can be observed in Table IX where the algorithms PSO, PSO-X, ABC, and ABC-X are better than BFO, BFO-A, and BFO-S algorithms and in Table X where the PSO and PSO-X are better than the other algorithms.

5. CONCLUSION

In this paper, we presented results verifying the differences in exploitation and exploration balance in different SI algorithms performing an unbiased comparison over three difficult structural engineering problems. The analysis of the statistical results indicated that no algorithm performed best for all the problems, except for the welded beam design problem. Notwithstanding, different results for each problem were achieved, suggesting that an algorithm can be the best for a problem but not for other one. This happens mainly due the different exploration/exploitation strategies of the algorithms, which direct the search in different ways. In addition, each problem has different fitness landscapes (represented by its cost function and its constraints), which, by itself, may lead the same algorithm to perform differently for each problem.

Such results are in accordance with the no free lunch theorem [43], which states that it is not possible to point out which is the best optimization algorithm without considering a specific problem. Generically speaking, the results showed that it is not possible to point which algorithm is more efficient for generic classes of problems, such as the constrained optimization problems approached here.

For real-world problems, both the quality of solution and the computational cost are important issues. In this work, the Pareto plot was found to be a useful tool for comparing the performance of algorithms considering such a multicriteria approach.

The explosion procedure used in PSO and ABC tend to increase significantly the number of function evaluations without, however, improving significantly the quality of solutions. This is contrary to what has been cited in the literature about this procedure. We speculate that the reason for this unexpected behavior is the parameters that regulate the frequency of explosion. Therefore, instead of using the standard values, a fine-tuning of such parameters can lead to better results.

On the basis of the results presented, we consider some combinations between algorithms as an alternative to solve these and other problems even more efficiently. Recent literature has indicated that the use of hybrid evolutionary systems working in a cooperative way can perform better than the use of single algorithms (for instance, [44–47]). A possible approach for this is to form a pipeline, passing the results from one algorithm to another or in parallel, choosing the best result from the parallel runs of several algorithms, or by having communication between the algorithms and each other. Therefore, we believe that hybrid/cooperative optimization strategies for highly constrained engineering problems are a promising future research.

ACKNOWLEDGEMENTS

The authors would like to thank UDESC (Santa Catarina State University) and FUMDES program for the financial support to R. S. Parpinelli as well as to the Brazilian National Research Council (CNPq) for research grant no. 309262/2007-0 to H. S. Lopes.

REFERENCES

1. Dorigo M, Stützle T. *Ant Colony Optimization*. The MIT Press: Cambridge, USA, 2004.
2. Kennedy J, Eberhart RC. *Swarm Intelligence*. Morgan Kaufmann: San Francisco, USA, 2001.
3. Clerc M. *Particle Swarm Optimization*. Wiley-ISTE Press: London, UK, 2006.
4. Parpinelli RS, Lopes HS. New inspirations in swarm intelligence: a survey. *International Journal of Bio-Inspired Computation* 2011; **3**(1):1–16.
5. Bonabeau E, Dorigo M, Theraulaz G. *Swarm Intelligence: from Natural to Artificial Systems*. Oxford University Press: New York, USA, 1999.
6. Garnier S, Gautrais J, Theraulaz G. The biological principles of swarm intelligence. *Swarm Intelligence* 2007; **1**(1):3–31.
7. Hazra J, Sinha AK. Environmental constrained economic dispatch using bacteria foraging optimization. In *Proceedings of International Joint Conference on Power System Technology and IEEE Power India Conference*. IEEE Press: Piscataway, USA, 2008; 1–6.
8. Das S, Biswas A, Dasgupta S, Abraham A. *Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications, Studies in Computational Intelligence*, Vol. 203: Berlin/Heidelberg, Germany, 2009. 23–55.
9. Passino K. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine* 2002; **22**(3):52–67.
10. Chen H, Zhu Y, Hu K. Self-adaptation in bacterial foraging optimization algorithm. *Proc. 3rd International Conference on Intelligent System and Knowledge Engineering*, Xiamen, China, 2008; 1026–1031.
11. Hembecker F, Godoy W, Jr., Lopes HS. Particle swarm optimization for the multidimensional knapsack problem. *Lecture Notes in Computer Science* 2007; **4331**:358–365.
12. Perlin HA, Lopes H, Mezzadri TC. Particle swarm optimization for object recognition in computer vision. *Lecture Notes in Computer Science* 2008; **5027**:11–21.
13. Poli R. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications* 2008; **2008**:1–10.
14. Benítez CMV, Lopes HS. A master–slave parallel genetic algorithm for protein structure prediction using the 3D-HP side-chain model. *Journal of the Brazilian of Computer Society* 2010; **16**(1):69–78.
15. Kalegari DH, Lopes HS. A differential evolution approach for protein structure optimization in a 2-D off-lattice protein model. *International Journal of Bio-Inspired Computation* 2010; **2**(3/4):242–250.
16. Karaboga D. An idea based on honey bee swarm for numerical optimization. *Technical Report*, Erciyes University, Engineering Faculty, Computer Engineering Department, Kayseri, Turkey, 2005.
17. Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation* 2009; **214**:108–132.
18. Akay B, Karaboga D. Parameter tuning for the artificial bee colony algorithm. *Lecture Notes in Artificial Intelligence* 2009; **5796**:608–619.
19. Baykasoğlu A, Özbakır L, Tapkan P. Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem. In *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*, Chan FTS, Tiwari MK (eds), 2007; 113–144.
20. Linh NT, Anh NQ. Application artificial bee colony algorithm (ABC) for reconfiguring distribution network. *Second International Conference on Computer Modeling and Simulation* 2010; **1**:102–106.
21. Karaboga D, Ozturk C. Neural networks training by artificial bee colony algorithm on pattern classification. *Neural Network World* 2009; **19**(3):279–292.
22. Marinakis Y, Marinaki M, Matsatsinis N. A hybrid discrete artificial bee colony—GRASP algorithm for clustering. *Proc. of International Conference on Computers and Industrial Engineering*, Troyes, France, 2009; 548–553.
23. Akay B, Karaboga D. Solving integer programming problems by using artificial bee colony algorithm. *Lecture Notes in Computer Science* 2009; **5883**:355–364.
24. Chidambaram C, Lopes HS. An improved artificial bee colony algorithm for the object recognition problem in complex digital images using template matching. *International Journal of Natural Computing Research* 2010; **1**(2):54–70.
25. Karaboga D, Ozturk C. A novel clustering approach: artificial bee colony (ABC) algorithm. *Applied Soft Computing* 2011; **11**:652–657.
26. Sabat SL, Udgata SK, Abraham A. Artificial bee colony algorithm for small signal model parameter extraction of MESFET. *Engineering Applications of Artificial Intelligence* 2010; **23**(5):689–694.
27. Goldberg DE. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional: Reading, MA, 1989.
28. Deb K, Srinivasan A. Monotonicity analysis, evolutionary multi-objective optimization, and discovery of design principles. *Technical Report KanGAL No. 2006004*, Indian Institute of Technology, Kanpur Genetic Algorithms Laboratory, Kanpur, India, 2006.
29. Ragsdell KM, Phillips DT. Optimal design of a class of welded structures using geometric programming. *ASME Journal of Engineering for Industries* 1976; **98**(3):1021–1025.
30. Deb K. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering* 2000; **186**(2–4):311–338.

31. Lee KS, Geem ZW. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering* 2005; **194**(36–38):3902–3933.
32. Fesanghary M, Mahdavi M, Minary-Jolandan M, Alizadeh Y. Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. *Computer Methods in Applied Mechanics and Engineering* 2008; **197**(33–40):3080–3091.
33. Mahdavi M, Fesanghary M, Damangir E. An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation* 2007; **188**(2):1567–1579.
34. Kannan BK, Kramer SN. An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design* 1994; **116**(2):405–411.
35. Deb K, Gene AS. A robust optimal design technique for mechanical component design. In *Evolutionary Algorithms in Engineering Applications*, Dasgupta D, Michalewicz Z (eds). Springer: Berlin, Germany, 1997; 497–514.
36. Coello CAC. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry* 2000; **41**(2):113–127.
37. Coello CAC. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems* 2000; **17**:319–346.
38. Khan MR, Willmert KD, Thornton WA. An optimality criterion method for large-scale structures. *AIAA Journal* 1979; **17**(7):753–761.
39. Rizzi P. Optimization of multi-constrained structures based on optimality criteria. *Proc. AIAA/ASME/SAE 17th Structures, Structural Dynamics and Materials Conference*, King of Prussia, Pennsylvania, 1976; 448–462.
40. Schmit LA, Miura H. Approximation concepts for efficient structural synthesis. *AIAA Journal* 1976; **12**(3):692–699.
41. Venkayya VB. Design of optimum structures. *Computers & Structures* 1971; **1**(1-2):265–309.
42. Hinkelmann K, Kempthorne O. *Design and Analysis of Experiments*, Vol. I and II. J. Wiley & Sons: London, UK, 2008.
43. Wolpert D, Macready WG. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1997; **1**(1):67–82.
44. Masegosa AD, Pelta DA, del Amo IG, Verdegay JL. On the performance of homogeneous and heterogeneous cooperative search strategies. In *Nature Inspired Cooperative Strategies for Optimization, Studies in Computational Intelligence*, Vol. 236, Krasnogor N., Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM, Pelta DA (eds). Springer: Berlin, Germany, 2009; 287–300.
45. Benítez CMV, Parpinelli RS, Lopes HS. Parallelism, hybridism and coevolution in a multi-level ABC-GA approach for the protein structure prediction problem. *Concurrency and Computation: Practice and Experience* 2011. DOI: 10.1002/cpe.1857.
46. Sun L, Yoshida S, Cheng X, Liang Y. A cooperative particle swarm optimizer with statistical variable interdependence learning. *Information Sciences* 2012; **186**:20–39.
47. Inthachot M, Supratid S. A multi-subpopulation particle swarm optimization: a hybrid intelligent computing for function optimization. *International Conference on Natural Computation* 2007; **5**:679–684.