Novel Fish Swarm Heuristics for Bound Constrained Global Optimization Problems

Ana Maria A.C. Rocha[1], Edite M.G.P. Fernandes[2]
and Tiago F.M.C. Martins[2]

[1] Department of Production and Systems, University of Minho,
4710-057 Braga, Portugal
arocha@dps.uminho.pt
[2] Algoritmi R&D Centre, University of Minho,
4710-057 Braga, Portugal
emgpf@dps.uminho.pt, martins.tiago41@gmail.com

Abstract. The heuristics herein presented are modified versions of the artificial fish swarm algorithm for global optimization. The new ideas aim to improve solution accuracy and reduce computational costs, in particular the number of function evaluations. The modifications also focus on special point movements, such as the random, search and the leap movements. A local search is applied to refine promising regions. An extension to bound constrained problems is also presented. To assess the performance of the two proposed heuristics, we use the performance profiles as proposed by Dolan and Moré in 2002. A comparison with three stochastic methods from the literature is included.

Keywords: Global optimization, Derivative-free method, Swarm intelligence, Heuristics

1 Introduction

In this paper, we consider the problem of finding a global solution of a nonlinear optimization problem with bound constraints in the following form:

$$\underset{x \in \Omega}{\text{minimize}} \, f(x) \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function and $\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\}$ is the feasible region. The objective function f may be non-smooth and may possess many local minima in the set Ω since we do not assume that f is convex. Many derivative-free algorithms and heuristics have been proposed to solve (1), namely those based on swarm intelligence. Probably the most well-known are the particle swarm optimization [11, 20], the ant colony [10, 15] and the artificial bee colony [9] algorithms. Evolutionary strategies are also common and new algorithms are always emerging [16]. Recently, an artificial life computing algorithm that simulates fish swarm behaviors was proposed and applied in some engineering context [7, 8, 18, 19]. The behavior of a fish swarm inside water is

beautiful to watch and although it is simple in concept, it turns out to be a complex system to simulate. Fish swarm movements seem randomly defined and yet they are objectively synchronized. Fishes desire to stay close to the swarm, protecting themselves from predators and looking for food, and to avoid collisions within the group. These behaviors inspire mathematical modelers aiming to solve optimization problems in an efficient manner. Behavioral model-based optimization algorithms seek to imitate, as well as to make variations on the swarm behavior in nature, and to create new types of abstract movements. The fish swarm behaviors inside water may be summarized as below:

i) *random* behavior - in general, fish swims randomly in water looking for food and other companions;
ii) *searching* behavior - this is a basic biological behavior since fish tends to the food; when fish discovers a region with more food, by vision or sense, it goes directly and quickly to that region;
iii) *swarming* behavior - when swimming, fish naturally assembles in groups which is a living habit in order to guarantee the existence of the swarm and avoid dangers;
iv) *chasing* behavior - when a fish, or a group of fishes, in the swarm discovers food, the others in the neighborhood find the food dangling quickly after it;
v) *leaping* behavior - when fish stagnates in a region, it leaps to look for food in other regions.

The artificial fish is a fictitious entity of a true fish. Its movements are simulations and interpretations of the above listed fish behaviors [8]. The environment in which the artificial fish moves, searching for the minimum, is the feasible search space of the minimization problem. Considering the problem that is addressed in the paper, the feasible search space is the set Ω (see Eq. (1)). The position of an artificial fish in the solution space is herein denoted by a point x (a vector in $\mathbb{R}^n$).

We will use the words 'fish' and 'point' interchangeably throughout the paper. The artificial fish swarm (AFS) algorithm uses a population of points to identify promising regions looking for a global solution [18]. This paper proposes new heuristics to incorporate into the AFS algorithm aiming to improve accuracy and reduce computational costs. The new heuristics are focused on:

− the algorithmic interpretation of some fish behaviors, like random, searching and leaping;
− a greedy criterion aiming to define a selecting behavior;
− a random local search, aiming to refine the best solution at the end of each iteration;
− a priority-based AFS strategy, aiming to speed fish movements.

We remark that the modified heuristics are devised to consider the bound constraints of the problem.

For a practical assessment of the proposed modifications, numerical experiments are carried out involving a set of 25 benchmark problems. The results show that our proposals have promising performances.

The organization of the paper is as follows. In Sect. 2, we introduce the general AFS paradigm. Sect. 3 introduces the proposed modifications and presents a detailed description of the procedures in the new algorithm. Sect. 4 describes the numerical experiments and Sect. 5 presents the conclusions.

2 The Artificial Fish Swarm Paradigm

The used notation is as follows: $x^i \in \mathbb{R}^n$ denotes the ith point of a population; x^{best} is the point that has the least objective function value and f_{best} is the corresponding function value; $x_k^i \in \mathbb{R}$ is the kth ($k = 1, \ldots, n$) component of the point x^i of the population; m is the number of points in the population.

The crucial issue of the artificial fish swarm algorithm is the 'visual scope' of each point. This represents the closed neighborhood of x^i with ray equal to a positive quantity v. In the context of the simple bound constrained problem (1), addressed in this paper, the definition of v is shown later on in Eq. (7).

Let I^i be the set of indices of the points inside the 'visual scope' of point x^i, where $i \notin I^i$ and $I^i \subset \{1, \ldots, m\}$, and let np^i be the number of points in its 'visual scope'. Depending on the relative positions of the points in the population, three possible situations may occur:

- when $np^i = 0$, the 'visual scope' is empty, and the point x^i, with no other points in its neighborhood to follow, moves randomly searching for a better region;
- when the 'visual scope' is crowded, the point has some difficulty in following any particular point, and searches for a better region choosing randomly another point (from the 'visual scope') and moves towards it;
- when the 'visual scope' is not crowded, the point is able either to swarm moving towards the central or to chase moving towards the best point.

The condition that decides when the 'visual scope' of x^i is not crowded is

$$\frac{np^i}{m} \leq \theta, \tag{2}$$

where $\theta \in (0, 1]$ is the crowd parameter. In this situation, point x^i has the ability to swarm or to chase. The algorithm simulates both movements and chooses the best in the sense that a better function value is obtained.

The swarming behavior is characterized by a movement towards the central point in the 'visual scope' of x^i, defined by

$$c = \frac{\sum_{j \in I^i} x^j}{np^i}. \tag{3}$$

However, the swarming behavior is activated only if the central point has a better function value than that of x^i. Otherwise, the point x^i follows the searching behavior.

In the searching behavior, a point is randomly chosen in the 'visual scope' and a movement towards it is carried out if the random point improves over x^i. Otherwise, the point moves randomly.

The chasing behavior is carried out when a point, denoted by $x^{\min}$, with the minimum function value inside the 'visual scope' of x^i, satisfies

$$f(x^{\min}) \equiv \min \left\{ f(x^j) : j \in I^i \right\} < f(x^i). \tag{4}$$

However, if this last condition is not satisfied then the point activates the searching behavior. We refer to [18, 19] for some details.

3   The Modified AFS Algorithm

First, we present the proposed main algorithm that incorporates the selecting and local behaviors. The algorithm has eight main procedures: *Initialize*, *Random*, *Search*, *Swarm*, *Chase*, *Select*, *Leap* and *Local*. Then, we present details of our proposals for the procedures to translate random, searching and leaping behaviors. Further, a simple procedure *Select*, aiming to define the elite population for the next iteration, and the procedure *Local* to refine the search around x^{best}, are also presented.

Later on in this section, another modification to the below main algorithm is introduced to speed fish movements.

We remark that the algorithm has been devised to solve bound constrained optimization problems in a way that feasibility is always maintained throughout all point movements. In the Algorithm 1, t represents the iteration counter.

Algorithm 1 *Modified AFS algorithm*
$t \leftarrow 0$
$x^i(t)(i = 1, \ldots, m) \leftarrow$ Initialize
<u>While</u> *stopping criteria are not met* <u>do</u>
    <u>For</u> *each $x^i(t)$* <u>do</u>
        <u>If</u> *'visual scope' is empty* <u>then</u>
            $y^i(t) \leftarrow$ Random*($x^i(t)$)*
        <u>else</u>
            <u>If</u> *'visual scope' is crowded* <u>then</u>
                $y^i(t) \leftarrow$ Search*($x^i(t)$)*
            <u>else</u>
                $y^i(t) \leftarrow$ *best of* Swarm*($x^i(t)$) and* Chase*($x^i(t)$)*
    <u>End for</u>
    $x^i(t+1)(i = 1, \ldots, m) \leftarrow$ Select*($x^i(t), y^i(t)(i = 1, \ldots, m)$)*
    <u>If</u> *'stagnation' occurs* <u>then</u>
        $x^{\text{rand}}(t+1) \leftarrow$ Leap*($x^{\text{rand}}(t+1)$)*
    $x^{\text{best}}(t+1) \leftarrow$ Local*($x^{\text{best}}(t+1)$)*
    $t \leftarrow t + 1$
<u>End while</u>

In this algorithm, x^{rand} represents a randomly selected point from the population. We now present details of each procedure. To simplify the notation, the dependence of each point on t is dropped out whenever the concerned entities are from the same iteration.

3.1 Initialize

The procedure *Initialize* aims to randomly generate the initial population of m points in the set Ω. Each point x^i in the population is componentwise computed by

$$x_k^i = l_k + \omega(u_k - l_k), \ \text{for} \ k = 1, \dots, n, \tag{5}$$

where u_k and l_k are the upper and lower bounds respectively of the set Ω, and ω is an independent uniform random number distributed in the range $[0, 1]$. The simplified notation $\omega \sim U[0, 1]$ will be used throughout the paper. The procedure computes the best and the worst function values found in the population as follows:

$$f_{\mathrm{best}} = \min\left\{f(x^i) : i = 1, \dots, m\right\} \ \text{and} \ f_{\mathrm{worst}} = \max\left\{f(x^i) : i = 1, \dots, m\right\}. \tag{6}$$

3.2 The 'visual scope'

To define the 'visual scope', a fixed value for the neighborhood ray, depending on the bound constraints of the problem, is defined as

$$v = \delta \max_{k \in \{1, \dots, n\}} (u_k - l_k), \tag{7}$$

where δ is a positive visual parameter. In general, this parameter is maintained fixed over the iterative process. However, experiments show that a slow reduction accelerates the convergence to the solution [4]. Thus, we use the following update $\delta = \max\{\delta_{\min}, \mu_\delta \delta\}$, every s iterations, where $0 < \mu_\delta < 1$ and $\delta_{\min}$ is a sufficiently small positive constant.

3.3 Search

When the 'visual scope' is crowded, see Eq. (2), the algorithm activates the procedure *Search*. Here, a point inside the 'visual scope' is randomly selected, x^{rand} (rand $\in I^i$), and the point x^i is moved towards it if the condition $f(x^{\mathrm{rand}}) < f(x^i)$ holds. Otherwise, the point x^i is moved randomly (see procedure *Random* below). When x^i is moved towards x^{rand}, the following direction is used $d^i = x^{\mathrm{rand}} - x^i$. This movement is carried out component by component $(k = 1, \dots, n)$ and takes into account the allowed movement towards the upper bound u_k and lower bound l_k of the set Ω. Furthermore, the direction of movement is normalized so that feasibility can be maintained. Algorithm 2 describes this simple movement along a specific direction d.

Algorithm 2 *Movement*
$\omega \sim U[0, 1]$
<u>For</u> *each component x_k* <u>do</u>
    <u>If</u> $d_k > 0$ <u>then</u>

$$y_k \leftarrow x_k + \omega \frac{d_k}{\|d\|}(u_k - x_k)$$

    <u>else</u>

$$y_k \leftarrow x_k + \omega \frac{d_k}{\|d\|}(x_k - l_k)$$

<u>End for</u>

3.4 Random

The procedure *Random* is used to move a point randomly inside the 'visual scope'. This procedure is called when the 'visual scope' is empty or when, in the procedure *Search*, the point x^{rand} is worst than x^i. Details of our interpretation of a random behavior are shown in the Algorithm 3.

Algorithm 3 *Random*
<u>For</u> *each component x_k* <u>do</u>
    $\omega_1 \sim U[0, 1]; \omega_2 \sim U[0, 1]$
    <u>If</u> $\omega_1 > 0.5$ <u>then</u>
        <u>If</u> $u_k - x_k > v$ <u>then</u>
            $y_k = x_k + \omega_2 v$
        <u>else</u>
            $y_k = x_k + \omega_2(u_k - x_k)$
    <u>else</u>
        <u>If</u> $x_k - l_k > v$ <u>then</u>
            $y_k = x_k - \omega_2 v$
        <u>else</u>
            $y_k = x_k - \omega_2(x_k - l_k)$
<u>End for</u>

3.5 Swarm and Chase

The procedures *Swarm* and *Chase* perform movements that can be considered as local searches. In fact, when the 'visual scope' of a point x^i is not crowded, the point may have two behaviors. One is related with a movement towards the central point of the 'visual scope', c, computed as shown in Eq. (3), denoted by swarming behavior. The procedure *Swarm* defines the direction of the movement as $d^i = c - x^i$ and x^i is moved according to the Algorithm 2 if $f(c) < f(x^i)$. Otherwise, the procedure *Search* is called.

The other, denoted by chasing behavior, is related with a movement towards the point that has the least function value, x^{min}, as previously defined in Eq. (4). Thus, the procedure *Chase* defines the direction, $d^i = x^{\text{min}} - x^i$, and moves x^i according to the movement defined in the Algorithm 2 if x^{min} improves over x^i. Otherwise, the procedure *Search* is called.

3.6 Select

The Algorithm 1 includes a selection task aiming to accept trial points only if they improve over the previous ones. Thus, the computed trial point $y^i(t)$ replaces $x^i(t)$, for the next iteration, if the greedy criterion holds:

$$x^i(t+1) = \begin{cases} y^i(t), & \text{if } f(y^i(t)) < f(x^i(t)) \\ x^i(t), & \text{otherwise} \end{cases} . \tag{8}$$

3.7 Leap

When the best objective function value in the population does not change for a certain number of iterations, the algorithm may have fallen into a local minimum. This is herein denoted by 'stagnation'. The other points of the population will in the subsequent iterations eventually converge to that local minimum. To be able to leap out the local and try to converge to the global minimum, the algorithm implements the procedure *Leap*, every r iterations, when the best solutions are not significantly different, i.e., when

$$|f_{\text{best}}(t) - f_{\text{best}}(t-r)| \leq \eta \tag{9}$$

holds, for a small positive tolerance η, where r defines the periodicity for testing the criterion. A point is randomly selected from the population and a random movement is carried inside the set Ω. The Algorithm 4 describes the pseudo-code of this procedure *Leap*. In the algorithm, x^{rand} represents a randomly selected point from the population (rand $\in \{1, \ldots, m\}$).

Algorithm 4 *Leap*
<u>For</u> *each component* x_k^{rand} <u>do</u>
    $\omega_1 \sim U[0,1]$; $\omega_2 \sim U[0,1]$
    <u>If</u> $\omega_1 > 0.5$ <u>then</u>
        $x_k^{\text{rand}} = x_k^{\text{rand}} + \omega_2 \left(u_k - x_k^{\text{rand}} \right)$
    <u>else</u>
        $x_k^{\text{rand}} = x_k^{\text{rand}} - \omega_2 \left(x_k^{\text{rand}} - l_k \right)$
<u>End for</u>

3.8 Local

The modified AFS algorithm includes a procedure aiming to gather the local information around the best point of the population. It is denoted by procedure *Local* and corresponds to a simple random line search applied component by component to x^{best}. The main steps are as follows. For each component k ($k = 1, \ldots, n$), x^{best} is assigned to a temporary point z. Next, a random movement of length $\nu \max_{k \in \{1,\ldots,n\}}(u_k - l_k)$, where ν is a small positive parameter, is carried out and if a better point is obtained within $L_{\max}$ iterations, x^{best} is replaced by z, the search ends for that component and proceeds to another one. Although a more sophisticated procedure could be used [14], this simple local search has been shown to improve accuracy at a reduced computational cost.

3.9 Stopping Criteria

The algorithm is terminated when one of the following conditions is verified:

$$nfe > nfe_{\max} \text{ or } |f_{\text{worst}} - f_{\text{best}}| < \varepsilon \tag{10}$$

where nfe represents the counter for the number of objective function evaluations, $nfe_{\max}$ is the maximum number of function evaluations allowed and ε is a small positive tolerance. The values f_{worst} and f_{best} were previously defined in Eq. (6).

3.10 A Priority-based AFS Strategy

The motivation for the below proposed modification is the following. When the 'visual scope' of a point x^i is not crowded, Algorithm 1 simulates two behaviors, the swarming and the chasing behaviors. To check if the movements towards the points $x^{\min}$ and c are carried out, their function values are compared with $f(x^i)$ and although $f(x^{\min})$ is already known, the objective function must be evaluated at c. Furthermore, the two computed trial points must be compared to each other to select the best one. This procedure is expensive in terms of function evaluations.

To reduce function evaluations, the proposal simulates one behavior at each time instead of trying both behaviors at the same time. We rank the chasing behavior with highest priority, so that the movement in direction to $x^{\min}$ is carried out first if $f(x^{\min}) < f(x^i)$. Otherwise, the swarming behavior will be the alternative. So, the movement in direction to c is then carried out if $f(c) < f(x^i)$. However, if the latter condition does not hold the procedure *Search* is then called. We denote this modification by mAFS-P.

4 Numerical Experiments

Twenty five small problems (with $2 \leq n \leq 10$), yet difficult to solve, from a benchmark set were used in our numerical experiments. The list is: ACK, BR, CB3, CB6, CM$_2$, EP, GP, GRP, GW, H3, H6, MC, NF2, NF3, OSP, PQ, RB, RG, S5, S7, S10, SBT, SF1, SF2 and WP, see Appendix B of [1]. The algorithms were coded in C#, and the results were obtained in a computer Intel Core 2 Duo P9700 2.8 GHz, with 6 GB 1066MHz of RAM, running Microsoft Windows 7.

First, the effect of some parameters on the algorithm performance is analyzed. Then, we compare the two new AFS heuristics: mAFS (as in Algorithm 1) and mAFS-P (with the movement towards $x^{\min}$ as the priority movement). Finally, we include a benchmark comparison with three stochastic-type solvers: (i) *ASA*, a point-to-point search based on adaptive simulated annealing [6]; (ii) *CMA-ES*, a population-based evolution strategy with a covariance matrix adaptation [5]; and (iii) *PSwarm*, a population-based particle swarm in a pattern search algorithm [17].

To compare computational requirement and solution accuracy, all the experiments are allowed to run until a specified maximum number of function evaluations is attained (see Eq. (10)). We use $nfe_{\max} = 100n^2$. The factor n^2 aims to show the effect of dimensionality on the algorithms performance, as higher dimension problems are in general more difficult to solve than lower dimension ones. Other user defined parameters are set as follows: $\varepsilon = 10^{-5}$, $\eta = 10^{-8}$ and $s = n$. In the procedure *Local* we set $\nu = 0.001$, and the maximum number of iterations therein allowed for the search along each component of the point is $L_{\max} = 10$. In all experiments, we solve each problem 30 times, and a population of $m = \min\{200, 10n\}$ points is used. The parameter r, from the procedure *Leap* is set equal to m.

4.1 Parameters Effect Using Factorial Design

Although no serious attempt was made to find the best parameter settings, some additional experiments were carried out to show the effect of parameter values on the performance of the modified AFS heuristics. Following a sensitivity study concerning the parameters δ, μ_δ and θ [4], we run mAFS-P using now a Design of Experiments approach [13]. A full factorial design based on two factors - the pair (initial δ, μ_δ) and θ - is implemented. The tested levels of each factor are:

- (initial δ, μ_δ) (4 levels): (1, 0.5), (1, 0.9), (n, 0.5), (n, 0.9);
- θ (3 levels): 0.5, 0.8, 1.

The factorial design carried out with the factors (initial δ, μ_δ) and θ requires 12 different combinations to be tested. Each combination was tested 30 times with different random seeds on all the previously referred 25 problems. The performance assessment is based on the average relative deviation (ARD) defined by:

$$\text{ARD} = \frac{1}{30} \sum_{l=1}^{30} 100 \frac{\left| f_{\text{best}}^l - f^* \right|}{|f^*|} \tag{11}$$

as suggested in [21], where f_{best}^l is the best solution found at run l and f^* is the global solution known in the literature, for a particular problem. The smaller the ARD the better the performance is. However, when $f^* = 0$, the average of the 30 best solutions is used, instead of the ARD in (11). The observed ARD values for the different combinations of levels of factors are statistically different. The 12 values of ARD obtained for eight selected problems (ACK with $n = 10$, BR with $n = 2$, CB6 with $n = 2$, GW with $n = 10$, H6 with $n = 6$, NF2 with $n = 4$, RG with $n = 10$, SBT with $n = 2$) are shown in Fig. 1. The plots in the figure show that the two sets of parameters affect the performance of the algorithm, and are dependent on the problem. There are however some tendencies: i) when $\mu_\delta = 0.9$, the value $\theta = 0.8$ gives better results, ii) when $\mu_\delta = 0.5$, then $\theta = 1$ gives better performances. The initial δ values, 1 and n, give equal performances. In the subsequent experiments we set $\mu_\delta = 0.9$, $\delta_{\min} = 0.1$, $\theta = 0.8$ and the initial δ to n.
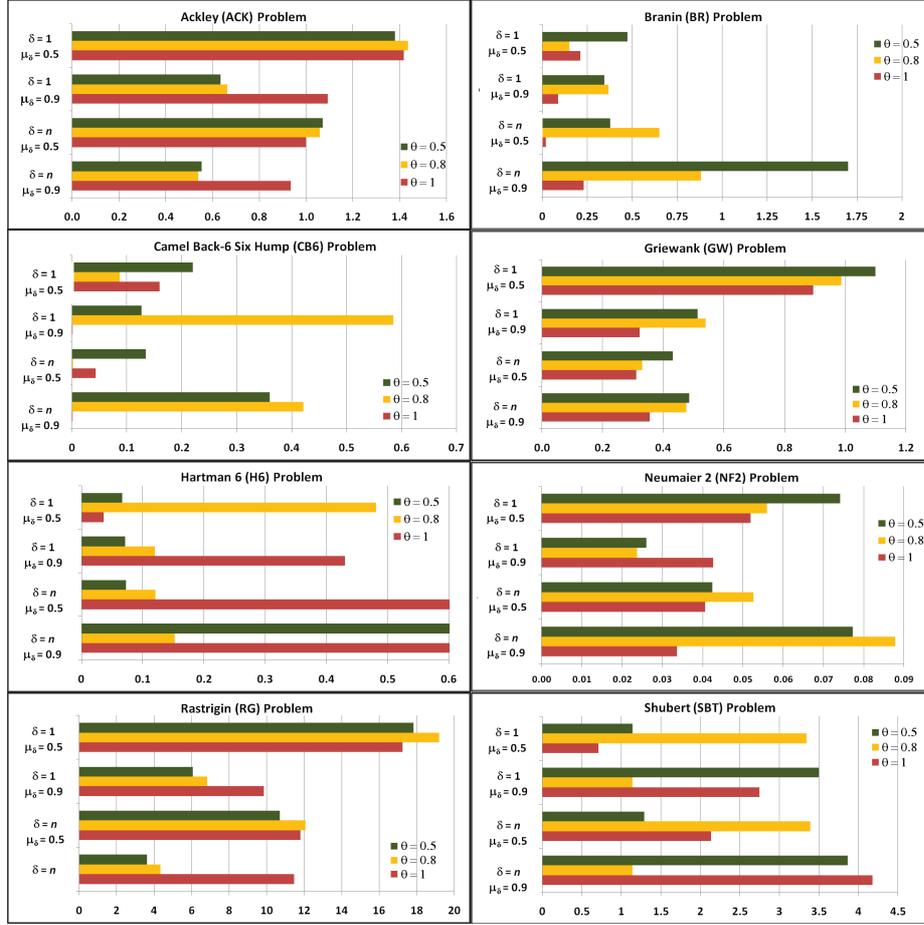
Fig. 1. Comparison of ARD with different (initial δ, μ_δ) and θ values.

4.2 Comparison Based on Performance Profiles

To compare the performance of the modified AFS algorithms, we use the performance profiles as described in Dolan and Moré's paper [3]. This is a recent and useful tool to interpret and visualize benchmark results [12]. Our profiles are mainly based on the metric f_{avg}, the average of the best solutions obtained over the 30 runs. Occasionally we use f_{best}, the best of the obtained solutions. It has been advised to report the central tendency of the results to measure and compare the performance of stochastic algorithms, since the best result of all is always biased and smaller than all the others [2].

Let $\mathcal{P}$ and $\mathcal{S}$ be the set of problems and the set of solvers in comparison, respectively, and $m_{p,s}$ be the performance metric used when solving problem $p \in \mathcal{P}$ by solver $s \in \mathcal{S}$. The relative comparison is based on the performance

ratios defined by

$$r_{p,s} = \begin{cases} 1 + m_{p,s} - \min\{m_{p,s} : s \in \mathcal{S}\}, \text{ if } \min\{m_{p,s} : s \in \mathcal{S}\} < \epsilon \\ \dfrac{m_{p,s}}{\min\{m_{p,s} : s \in \mathcal{S}\}}, \text{ otherwise} \end{cases}, \qquad (12)$$

for $\epsilon = 0.00001$ [17]. The overall assessment of the performance of a particular solver s is given by

$$\rho_s(\tau) = \frac{\text{no. of problems where } r_{p,s} \leq \tau}{\text{total no. of problems}}. \qquad (13)$$

Thus, $\rho_s(\tau)$ gives the probability, for solver $s \in \mathcal{S}$, that $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio. The value of $\rho_s(1)$ gives the probability that the solver s will win over the others in the set. Thus, to just see which solver is the best, i.e., which solver has the least value of the performance metric mostly, then $\rho_s(1)$ should be compared for all the solvers. The higher the ρ_s the better the solver is. On the other hand, $\rho_s(\tau)$ for large values of τ measures the solver robustness.

Comparing mAFS and mAFS-P. Here we aim to compare the two novel AFS heuristics: mAFS and mAFS-P. Figure 2 contains two plots with the performance profiles obtained when $100n^2$ function evaluations are allowed.
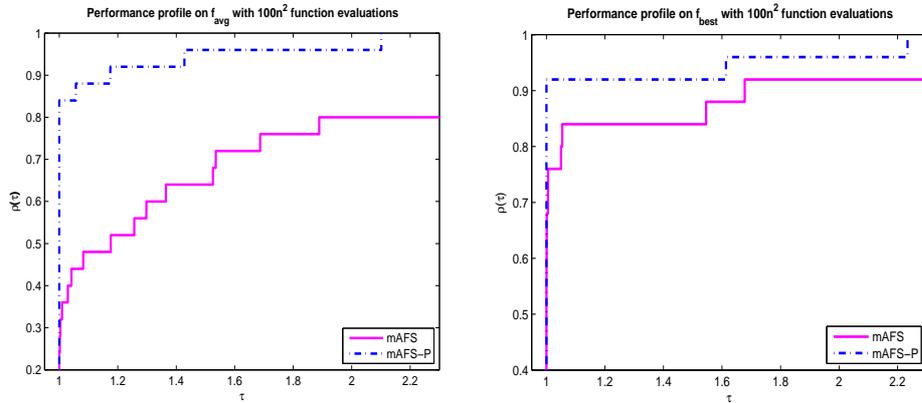

Fig. 2. Performance profiles on f_{avg} and f_{best} with $100n^2$ function evaluations.

From the plot on the left, based on the average performance, we conclude that the mAFS-P outperforms mAFS in 85% of the tested problems. This means that in 85% of the problems the values of f_{avg} - the metric $m_{p,s}$ in these profiles - obtained by mAFS-P are better or equal to those obtained by mAFS. Their corresponding performance ratios $r_{p,s}$ are then equal to one (see Eq. (12)). We

may conclude that when the allowed number of function evaluations is small, the mAFS-P version is able to reach, in average, the most accurate solutions. The plot on the right shows the profiles based on f_{best}. The version mAFS-P still gives the best solutions for most of the problems.

We also run both heuristics using $nfe_{\max} = 1000n^2$. Figure 3 shows the profile based on the metric f_{avg}. The plot here has two parts. One aims to give more visibility near $\tau = 1$ and the other aims to show the tendency for large values of τ. When allowing a large number of function evaluations in each run, the heuristic mAFS-P reaches, in average, more accurate solutions than mAFS in about 68% of the problems.
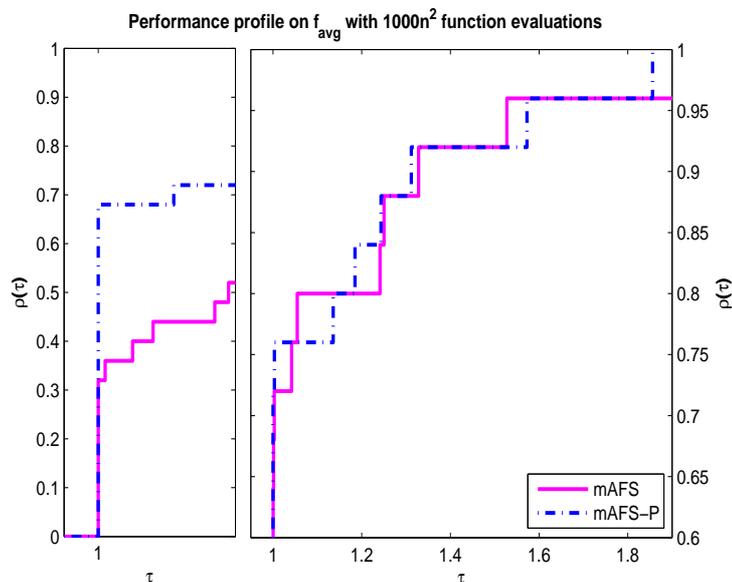
Fig. 3. Comparison of AFS heuristics mAFS and mAFS-P with $1000n^2$ function evaluations: performance profiles on f_{avg}

Comparison with other Solvers. In this part, we make a relative comparison between the heuristic mAFS-P and the three stochastic solvers *ASA*, *CMA-ES* and *PSwarm*. We run all the solvers until a specified maximum number of function evaluations is reached. Here we set $nfe_{\max} = 1000n^2$. Each problem is solved 30 independent times. The size of the population m is kept the same for all solvers. All the other parameters are set as the default values in the corresponding solvers. Usually they correspond to values that give the best results for most of the therein tested problems. The profiles are based on the metric f_{avg} and their relative performances are shown in Fig. 4.
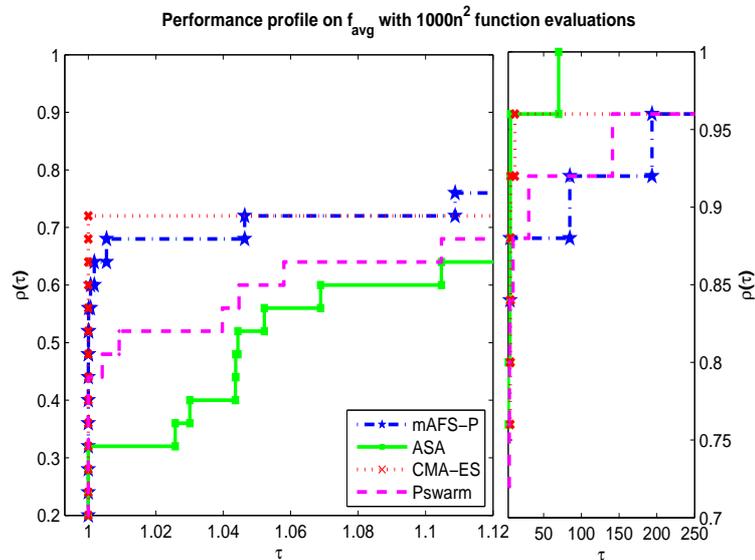
Fig. 4. Comparison of solvers based on performance profiles on f_{avg} with $1000n^2$ function evaluations.

We may conclude that *CMA-ES* outperforms the other solvers, followed by mAFS-P. While *CMA-ES* gives the best f_{avg} values for 72% of the tested problems, our heuristic mAFS-P gives the same best f_{avg} values in 60% of the problems, clearly superior to the other two solvers in comparison. Thus, the experiments show that the proposed mAFS-P algorithm has a promising performance.

5 Conclusions

This paper presents two novel heuristics, herein denoted by mAFS and mAFS-P, that rely on artificial life computing and swarm intelligence behaviors to detect promising regions and converge to the solution of global optimization problems. The modifications were introduced into the Artificial Fish Swarm algorithm aiming to improve solution accuracy and reduce computational efforts, namely the number of function evaluations. The new heuristics have been devised to handle bound constrained optimization problems. The new proposals for the heuristics were implemented and tested with a benchmark set of global optimization problems. A comparison with other stochastic solvers from the literature is also included. The herein proposed heuristics are effective in reaching the global solutions. The implementation of an augmented Lagrangian methodology in the heuristic mAFS-P to handle equality and inequality constraints is now under investigation.

References

1. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization, 31, 635–672 (2005)
2. Birattari, M., Dorigo, M.: How to assess and report the performance of a stochastic algorithm on a benchmark problem: mean or best result on a number of runs?, Optimization Letters, 1, 309–311 (2007)
3. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles, Mathematical Programming, 91, 201–213 (2002)
4. Fernandes, E.M.G.P., Martins, T.F.M.C., Rocha, A.M.A.C.: Fish swarm intelligent algorithm for bound constrained global optimization, In: Aguiar, J.V. (ed.), CMMSE 2009, 461–472 (2009) ISBN: 978-84-612-9727-6
5. Hansen, N.: The CMA evolution strategy: a comparing review, In: Lozano, J.A., Larranaga, P., Inza, I., Bengoetxea, E. (eds.), Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pp. 75–102 (2006)
6. Ingber, L.: Adaptive simulated annealing (ASA): lessons learned, Control and Cybernetics, 25, 33–54 (1996)
7. Jiang, M., Mastorakis, N., Yuan, D., Lagunas, M.A.: Image segmentation with improved artificial fish swarm algorithm, In: Mastorakis, N., Mladenov, V., Kontargyri, V.T. (eds.) ECC 2008, Lecture Notes in Electrical Engineering, 28, pp. 133–138, Springer-Verlag (2009) ISBN: 978-0-387-84818-1
8. Jiang, M., Wang, Y., Pfletschinger, S., Lagunas, M.A., Yuan, D.: Optimal multiuser detection with artificial fish swarm algorithm, In: Huang, D.-S., Heutte, L., Loog, M. (eds.), CCIS 2, ICIC 2007, Springer-Verlag, 1084–1093 (2007)
9. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization, 39, 459–471 (2007)
10. Karimi, A., Nobahari, H., Siarry, P.: Continuous ant colony system and tabu search algorithms hybridized for global minimization of continuous multi-minima functions, Computational Optimization and Applications, 45, 639–661 (2010)
11. Kennedy, J., Eberhart, R.C.: Particle swarm optimization, IEEE International Conference on Neural Network, 1942–1948 (1995)
12. Mittelmann, H.D., Pruessner, A.: A server for automated performance analysis of benchmarking data, Optimization Methods and Software, 21, 105–120 (2006)
13. Montgomery, D.C.: Design and Analysis of Experiments, John Wiley & Sons (5th ed.) (2002)
14. Rocha, A.M.A.C., Fernandes, E.M.G.P.: Hybridizing the electromagnetism-like algorithm with descent search for solving engineering design problems, International Journal of Computer Mathematics, 86, 1932–1946 (2009)
15. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains, European Journal of Operational Research, 185, 1155–1173 (2008)
16. Stanoyevitch, A.: Homogeneous genetic algorithms, International Journal of Computer Mathematics, 87, 476–490 (2010)
17. Vaz, A.I.F., Vicente, L.N.: A particle swarm pattern search method for bound constrained global optimization, Journal of Global Optimization, 39, 197–219 (2007)

18. Wang, C.-R., Zhou, C.-L., Ma, J.-W.: An improved artificial fish-swarm algorithm and its application in feed-forward neural networks, In: Proceedings of the 4th ICMLC, pp. 2890–2894 (2005)
19. Wang, X., Gao, N., S. Cai, Huang, M.: An artificial fish swarm algorithm based and ABC supported QoS unicast routing scheme in NGI, In: Min, G. et al. (eds.) ISPA 2006, LNCS, vol. 4331, pp. 205–214. Springer-Verlag (2006)
20. Zahara, E., Hu, C.-H.: Solving constrained optimization problems with hybrid particle swarm optimization, Engineering Optimization, vol. 40, no. 11, pp. 1031-1049 (2008)
21. Zhang, C., Ning, J., Ouyang, D.: A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem, Computers & Industrial Engineering, 58, 1–11 (2010)