We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Bioluminescent Swarm Optimization Algorithm

Daniel Rossato de Oliveira[1], Rafael S. Parpinelli[2] and Heitor S. Lopes[3]
[1,2,3]*Bioinformatics Laboratory, Federal University of Technology Paraná (UTFPR), Curitiba (PR), 80230-901*
[2]*Department of Computer Science, Santa Catarina State University (UDESC), Joinville (SC), 89223-100*
Brazil

1. Introduction

Evolutionary Computation (EC) is a research area of metaheuristics mainly applied to real-world optimization problems. EC is inspired by biological mechanisms such as reproduction, mutation, recombination, natural selection and collective animal behavior. Two branches of EC can be highlighted: Evolutionary Algorithms (EA) comprising Genetic Algorithms (Goldberg, 1989), Genetic Programming (Koza, 1992), Differential Evolution (Storn & Price, 1997), Harmony Search (Geem et al., 2001), and others; and Swarm Intelligence (SI) comprising Ant Colony Optimization (ACO) (Dorigo & Stützle, 2004) and Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 2001; Poli et al., 2007) and others. The ACO metaheuristic[1] is inspired by the foraging behavior of ants. On the other hand, the PSO metaheuristic[2] is motivated by the coordinated movement of fish schools and bird flocks. Both ACO and PSO approaches have been applied successfully in a vast range of problems (Clerc, 2006; Dorigo & Stützle, 2004).

In recent years, new SI algorithms were proposed. They have in common biological inspirations, such as bacterial foraging (Passino, 2002), slime molds life cycle (Monismith & Mayfield, 2008), various bees behaviors (Karaboga & Akay, 2009), cockroaches infestation (Havens et al., 2008), mosquitoes host-seeking (Feng et al., 2009), bats echolocation (Yang, 2010), and fireflies bioluminescense (Krishnanand & Ghose, 2005; 2009; Yang, 2009).

This work proposes a new swarm-based evolutionary approach based on the bioluminescent behavior of fireflies, called Bioluminescent Swarm Optimization (BSO) algorithm. The BSO uses two basic characteristics of the Glow-worm Swarm Optimization (GSO) algorithm proposed by (Krishnanand & Ghose, 2005): the luciferin attractant, and the stochastic neighbor selection. However, BSO goes further introducing new features such as: stochastic adaptive step sizing, global optimum attraction, leader movement, and mass extinction. Besides, the proposed algorithm is hybridized with two local search techniques: local unimodal sampling and single-dimension perturbation. All these features makes BSO a powerful algorithm for hard optimization problems.

Experiments were done to analyze the sensitivity of the BSO to control parameters. Later, extensive experiments were performed using several benchmark functions with high

[1] ACO repository: http://iridia.ulb.ac.be/~mdorigo/ACO/
[2] PSO Repository: http://www.particleswarm.info

dimensionality and different degrees of complexity. The performance of the BSO was compared with a well-known SI method, Particle Swarm Optimization (PSO).
The remainder of the chapter is organized as follows. Section 2 reports the existent bioluminescence inspired algorithms, focusing on the Glowworm Swarm Optimization algorithm (GSO), in which our proposed algorithm is mainly based. Section 3 shows in details the BSO algorithm and its main differences to the GSO algorithm. Section 4 explain how our experiments were done, as well as the parameter tuning process that lead to default parameter values. The results of these experiments are shown and discussed in Section 5. Conclusions and future works are presented in Section 6.

2. Bioluminescense inspired algorithms

Lampyridae is a family of insects (order *Coleoptera*) that are capable to produce natural light (bioluminescense) to attract a mate or a prey. They are commonly called fireflies or lightning bugs. In the species *Lampyris noctiluca* the fireflies are also known as glow-worms and, despite of the name, they are not worms. In this species, it is always the female who glows, and only the male has wings. In other species, *Luciola lusitanica*, both male and female firefly may emit light and both have wings (Fraga, 2008; Shimomura, 2006).
If a firefly is hungry or looks for a mate, its light glows to make the attraction of insects or mates more effective. The brightness of the bioluminescent light depends on the available amount of a pigment called *luciferin*[3].
Two optimization algorithms reported in the literature were inspired by the bioluminescent behavior of *Lampyridae* insects. The Firefly Algorithm (FA) was proposed by (Yang, 2009) as a general purpose optimization algorithm and it was applied to the optimization of benchmark functions. The Glow-worm Swarm Optimization Algorithm (GSO) that was designed to capture multiple peaks in multimodal functions, without the aim of finding the global best.

2.1 Glow-worm Swarm Optimization algorithm

The Glow-worm Swarm Optimization (GSO) algorithm was first presented by (Krishnanand & Ghose, 2005) to model the collective behavior in robotics. In this algorithm, each glow-worm uses a probabilistic mechanism to select a neighbor that has an associated luciferin value, and moves towards it. Glow-worms are attracted to neighbors that glow brighter. The movements are based only on local information and interactions with selected neighbor. This enables the swarm of glow-worms to divide themselves into disjoint subgroups that eventually converge to multiple local optima of a given multimodal function. The GSO algorithm is memoryless and the glow-worms do not retain any information about the search space.
The original GSO is shown in Algorithm 1. It starts by randomly placing in the search space a population of n glow-worms of dimension d. Each solution $\vec{x}_i = [x_{i1}, x_{i2}, ..., x_{id}]$ is evaluated by a fitness function $f(\vec{x}_i)$, $i = 1, ..., n$. At the beginning, all the glow-worms contain the same amount of luciferin l_0 and the same neighborhood range decision r_0. Each iteration consists of a luciferin update phase, followed by a movement phase based on a transition rule. Other running parameters of the algorithm are: the luciferin decay constant (ρ), the luciferin enhancement constant (γ), the step size (s), the number of neighbors (n_t), the sensor range (r_s), and a constant value (β). However, the authors observed that only two parameters have significant influence in the behavior of the algorithm: n and r_s.

[3] See the UK Glow worm survey home page at: http://www.glowworms.org.uk

Set parameters: n, l_0, r_0, ρ, γ, β, s, r_s, n_t
Randomly generate the population of glow-worms $\vec{x}_i$
for $i = 1$ **to** n **do**
  Initialize luciferin $l_i(0) = l_0$
  Initialize neighborhood range $r_d^i(0) = r_0$
end for
$t = 1$
while stop condition not met **do**
  for each glow-worm i **do** $\{update\ luciferin\}$
    $l_i(t+1) = (1-\rho) \cdot l_i(t) + \gamma \cdot f(x_i(t))$
  end for
  for each glow-worm i **do** $\{movement\ phase\}$
    Find neighbors $N_i(t)$
    for each glow-worm $j \in N_i(t)$ **do**
      Compute probability $P_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}$
    end for
    Select glow-worm j using P_{ij}
    Update glow-worm position with $x_i(t+1) = x_i(t) + s[\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}]$
    Update decision range:
      $r_d^i(t+1) = min\{r_s,\ max\{0,\ r_d^i(t) + \beta \cdot (n_t - |N_i(t)|)\}\}$
  end for
  $t = t + 1$
end while

Algorithm 1: The Glow-worm Swarm Optimization Algorithm (GSO)

The GSO algorithm was used by (Krishnanand & Ghose, 2009) to find multiple optima in multimodal benchmark functions. This work also conducted detailed parameter tuning experiments. When compared with a niched-PSO, GSO showed better results concerning the number of peaks found in almost all test functions. GSO was also applied to hazard sensing in ubiquitous environments (Krishnanand & Ghose, 2008). It should be stressed that in all applications the objective is to find multiple peaks of multimodal functions.

3. The Bioluminescent Swarm Optimization algorithm

In this section we present in detail the proposed swarm-inspired algorithm for global optimization named the Bioluminescent Swarm Optimization (BSO) algorithm. The section is divided into five subsections, namely, a general description of BSO, stochastic adaptive step sizing, global optimum attraction, mass extinction, and local search procedures.

3.1 General description of BSO algorithm

The BSO algorithm, shown in Algorithm 2, can be loosely seen as a hybrid between PSO and GSO, but with some unique features. As GSO and many other algorithms, the first step is initializing n particles in the d-dimensional search space. All particles, defined by $\vec{x}_i = [x_{i1}, x_{i2}, ..., x_{id}]$, are evaluated by a fitness function $fit(\vec{x}_i)$, $i = 1, ..., n$. BSO uses luciferin-based attraction instead of fitness-based attraction between the particles, as

proposed by the GSO. This process is controlled by the parameters ρ and γ, which are the luciferin decay constant and the luciferin enhancement constant, respectively. It also uses stochastic step size, similarly to PSO, instead of fixed step as in the GSO. This step size also varies for each particle, according to its luciferin value, and controlled by the c_s parameter. This is shown in line 18 of Algorithm 2, later explained in Section 3.2.

1: Set parameters: n, ρ, γ, s_0, c_g, c_s, lR, eT
2: Randomly generate the bioluminescent particle population $\vec{x}_i$
3: **for** $i = 1$ **to** n **do**
4: Initialize luciferin $l_i(0) = 0$
5: **end for**
6: Find the global best $g(t)$
7: $t = 1$
8: **while** stop condition not met **do**
9: **for** each particle i **do** $\{update\ luciferin\}$
10: $l_i(t+1) = (1-\rho) \cdot l_i(t) + \gamma \cdot f(x_i(t))$
11: **end for**
12: **for** each glow-worm i **do** $\{movement\ phase\}$
13: Find neighbors $N_i(t)$
14: **for** each particle $j \in N_i(t)$ **do**
15: Compute probability $P_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}$
16: **end for**
17: Select glow-worm j using P_{ij}
18: Update particle step size with $s = s_0 \cdot \frac{1}{1 + c_s \cdot l_i(t)}$
19: Update glow-worm position with
$x_i(t+1) = x_i(t) + rand \cdot s \cdot [\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}] + c_g \cdot rand \cdot s \cdot [\frac{g(t) - x_i(t)}{\|g(t) - x_i(t)\|}]$
20: Find the global best $g(t)$
21: **if** $t\%lR = 0$ **then** $\{LocalSearchProcedures\}$
22: Perform strong local search on $g(t)$
23: **else**
24: Perform weak local search on $g(t)$
25: **end if**
26: **if** iterations without a new $g(t) = eT$ **then** $\{MassExtinction\}$
27: Reinitialize all particles but g(t)
28: **end if**
29: **end for**
30: $t = t + 1$
31: **end while**

Algorithm 2: The Bioluminescent Swarm Optimization Algorithm (BSO)

While the concept of global optimum does not exist in the GSO algorithm, every particle in BSO is attracted to the global optimum, like PSO. This attraction is part of the equation in line 19 of the Algorithm 2, controlled by the parameter c_g, and discussed later in Section 3.3. We also propose a mass extinction mechanism shown in line 27. It is controlled by the parameter eT, and explained in Section 3.4.

In the GSO, particles without neighbors, that is, particles that are the best in its vicinity, do not move. It is assumed that the most promising regions of the search space are those regions in which the best solutions were found and, therefore, they should be deeply explored. Consequently, we propose two local search schemes. The first it a weak local search, shown in line 24, and meant to be the default movement for the global best. The second one is a strong local search, shown in line 22. This procedure is supposed to take place at each lR iterations. Both methods will be discussed in Section 3.5.

A weakness of the GSO algorithm is the computational overhead due to frequent distance measurements. It is necessary to know the distance between each particle at each iteration. Therefore, the number of distance computations at each iteration is $\frac{n^2-n}{2}$, where n is the number of particles. This means that the time spent only in this phase of the algorithm varies quadratically with the population size. Consequently, this fact leads to a significant overhead when using large populations.

To avoid this problem, BSO considers an infinite radius for the neighborhood. This means that the neighborhood of a given particle is composed by all other particles having luciferin value larger than its own. Experiments reported in Section 4 showed that this approach leads to a huge improvement in processing time, with almost no degradation of the quality of solutions. Consequently, the BSO algorithm has four more parameters: c_s to control the adaptive step sizing, c_g, to control the global best attraction, eT, to control the mass extinction, and lR, to control the strong local search. On the other hand, it does not have the parameters β, r_s, r_0, and n_t, present in the GSO algorithm. This is due to the infinite radius adopted by the BSO for the neighborhood of each particle, since these four parameters were used to control the radius of each particle.

3.2 Stochastic adaptive step sizing

The BSO algorithm uses Equation 1 to compute the next position of a given particle:

$$x_i(t+1) = x_i(t) + rand \cdot s \cdot [\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}] + c_g \cdot rand \cdot s \cdot [\frac{g(t) - x_i(t)}{\|g(t) - x_i(t)\|}] \quad (1)$$

where $x_i(t)$ is the current particle position, $rand$ is a random number in the [0, 1] interval, s is the current step size of the particle, c_g is the global best attraction constant, and $g(t)$ is the global best position.

Unlike GSO that uses a fixed step size, BSO changes the step size stochastically, in the same way as in PSO. Besides, the maximum step is adaptive, according to Equation 2:

$$s = s_0 \cdot \frac{1}{1 + c_s \cdot l_i(t)} \quad (2)$$

where s_0 is the maximum step, $l_i(t)$ is the amount of luciferin of the particle, and c_s is a slowing constant. This means that a particle with a high level of luciferin will move slower, while a particle with a low level of luciferin will move faster. Since the level of luciferin is an information related to the fitness of the current and previous positions, particles laying in promising regions of the search space will perform a more thorough search, with small steps. Conversely, particles laying in regions of low quality will move faster, searching for better regions.

The step sizes resulting from Equation 2 must be between 0 and s_0. Therefore, one must assure that $l_i(t)$ is always non-negative, what can be achieved by guaranteeing non-negative fitness.

This is also needed due to the roulette-wheel selection method, which will be explained further.
The parameter c_s is the slowing constant. It adjusts how much a particle will slow down due to its luciferin value. Large values of c_s lead to particles with low luciferin levels to perform a thorough search, while small c_s values makes only particles with very high luciferin levels to perform this search.

3.3 Global optimum attraction

Although the concept of global optimum does not exist in the GSO, each particle in the BSO algorithm is attracted both to the selected neighbor and to the current global optimum, as in the PSO. This is shown in the third term of the Equation 1, where the constant c_g means the force of this attraction. This constant is usually very low, but the results are still noticeably affected. Experiments reported on Section 4 show the effects.

3.4 Mass extinction

The GSO algorithm usually shows a fast convergence to multiple local optima. In the BSO, the goal is to provide global optimization, not multiple peak finding. Although BSO usually shows a slow and smooth convergence, as reported in Section 5, an early stagnation may occur. Therefore, it is important to detect stagnation as soon as possible and take corrective measures to avoid it. This is done through a mechanism known as Mass Extinction, Decimation and Hot-Boot or, simply, Explosion. This method is frequently used in PSO, genetic algorithms and other EC algorithms (Benítez & Lopes, 2010; Hembecker et al., 2007; Kalegari & Lopes, 2010), although not used in the original GSO.
Explosion works by reinitializing all or part of the particles, but keeping the main information gathered so far, like global and local optima. In the BSO, since there is no intrinsic local optimum information, we maintain just the best particle. This occurs when there is no improvement of the best solution for a given number of iterations, represented by the eT parameter. In the Algorithm 2, this verification is done in line 26. If the condition is true, that is, the last improvement of the global best $g(t)$ took place more than eT iterations before, all particles are reinitialized, except $g(t)$, as shown in line 27.

3.5 Local search procedures

Due to the dynamic grouping nature of the BSO particles, this algorithm performs strong exploration. However, experiments have shown that its exploitation capability is not intrinsically good. We propose a corrective measure using two local search procedures: a weak one, to be executed on the best individual at each iteration; and a strong one, to be executed on the global best found so far, at each lR iterations. In our experiments, Local Unimodal Sampling (LUS) (Pedersen, 2010) was used every iteration, and a Single-Dimension Perturbation Search (SDPS) at each lR iterations. The difference between these local search procedures is not on quality or overhead of the algorithm, but only the computational effort applied in each one. The strong one is meant to use much more function evaluations than the weak.
Since the neighborhood of a particle is composed by the particles with higher luciferin levels than itself, the best particle has no neighbors and, thus, does not move. Spending computational effort with the worse particles, while leaving the best one behind is not interesting, and so the weak local search is meant to correct this. Therefore, the weak local

search is the default movement for the best particle. It is allowed one turn off the strong local search, but the weak local search is an essential part of the BSO algorithm.

3.5.1 Local Unimodal Sampling

The Local Unimodal Sampling (LUS), shown in Algorithm 3, was proposed by (Pedersen, 2010) as a simple and fast method for numerical optimization that is adequate to unimodal search spaces. It works by randomly sampling a position within a radius from the base position, and decreasing this radius exponentially. Its parameters are the initial position $\vec{x_0}$, initial sampling radius r_{0w}, the decrease rate q, and the iteration limit n_w.

Set parameters: $n_w, r_{0w}, q, \vec{x_0}$
$\vec{x} = \vec{x_0}$
$r = r_{0w}$
for $i = 1$ **to** n_w **do**
  Randomly generate the movement vector $\vec{a}$ within r of $\vec{x}$
  if $f(\vec{x} + \vec{a}) > f(\vec{x})$ **then**
    $\vec{x} = \vec{x} + \vec{a}$
  else
    $r = q \cdot r$
  end if
end for

Algorithm 3: Local Unimodal Sampling (LUS) algorithm.

3.5.2 Single-Dimension Perturbation Search

The strong local search procedure (SDPS), shown in Algorithm 4, is just a random perturbation in a single random dimension in each iteration. It is similar to the LUS, but the main differences are: single-dimension instead of multi-dimension movement, and linear decay of search space radius, instead of exponential. Its parameters are the initial position $\vec{x_0}$, initial sampling radius r_{0s}, and the iteration limit n_s.

Set parameters: n_s, $r_{0s}, \vec{x_0}$
$\vec{x} = \vec{x_0}$
for $i = 1$ **to** n_s **do**
  $\vec{c} = \vec{x}$ $\{CandidateSolution\}$
  $r = r_{0s} \cdot \frac{n_s - i}{n_s}$
  Randomly choose a dimension j to perturbate
  Compute $step = rand \cdot r$ $\{rand \in [-1, 1]\}$
  $\vec{c}[j] = \vec{c}[j] + step$
  if $f(\vec{c}) > f(\vec{x})$ **then**
    $\vec{x} = \vec{c}$
  end if
end for

Algorithm 4: Single-Dimension Perturbation Search (SDPS)

4. Experiments

All experiments were done using a desktop computer with a 2.8GHz quad-core processor, 2GB of RAM, running a minimal installation of Arch Linux. The application software was development ANSI-C programming language. Since the BSO is a stochastic algorithm, all

experiments were repeated 100 times with different random seeds. The performance of each approach takes into account the average best solution found in each run and the average processing time.

The BSO algorithm was applied to four well-known benchmark functions, extensively used in the literature for the evaluation of metaheuristics (Digalakis & Margaritis, 2002). The definition of the functions, the corresponding range of values and the minimum value known are shown in Table 1.

Function	Ranges	Minimum value
$f_1(\vec{x}) = \sum_{i=1}^{d}(x_i^2 - 10\cos(2\pi x_i) + 10)$	$-5.12 \leq x_i \leq 5.12$	$f_1(\vec{0}) = 0$
$f_2(\vec{x}) = \frac{1}{4000}\left(\sum_{i=1}^{d} x_i^2\right) - \left(\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + 1$	$-600 \leq x_i \leq 600$	$f_2(\vec{0}) = 0$
$f_3(\vec{x}) = \sum_{i=1}^{d-1}\left(0.5 + \frac{\sin^2\left(\sqrt{x_{i+1}^2 + x_i^2}\right) - 0.5}{\left(0.001\left(x_{i+1}^2 + x_i^2\right) + 1\right)^2}\right)$	$-100 \leq x_i \leq 100$	$f_3(\vec{0}) = 0$
$f_4(\vec{x}) = \sum_{i=1}^{d-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$-30 \leq x_i \leq 30$	$f_3(\vec{1}) = 0$

Table 1. Numerical benchmark functions

The first function ($f_1(\vec{x})$) is the Rastrigin function that is a multimodal function and is based on the Sphere function, with the addition of cosine modulation to produce many local minima. The locations of the minima are regularly distributed. The function was originally proposed for two dimensions, and later generalized to d dimensions (Mühlenbein et al., 1991). The main difficulty in finding optimal solutions to this function is that an optimization algorithm can be easily trapped in a local optimum on its way towards the global optimum. $\vec{x}$ is defined in the range of $[-5.12, 5.12]$ and the global minimum value for $f_1(\vec{x})$ is 0 and the corresponding global optimum solution is $\vec{x}_{opt} = (x_1, x_2, \ldots, x_d) = (0, 0, \ldots, 0)$.

The second function ($f_2(\vec{x})$) is the Griewank function (Griewank, 1981) that is strongly multimodal, because the number of local optima increases exponentially with the dimensionality (Cho et al., 2008). $\vec{x}$ is defined in the range of $[-600, 600]$ and the global minimum value for $f_2(\vec{x})$ is 0 and the corresponding global optimum solution is $\vec{x}_{opt} = (x_1, x_2, \ldots, x_d) = (0, 0, \ldots, 0)$.

The third function ($f_3(\vec{x})$) is the generalized Schaffer function F6 (Floudas & Pardalos, 1990) that is also strongly multimodal. $\vec{x}$ is defined in the range of $[-100, 100]$ and the global minimum value for $f_3(\vec{x})$ is 0 and the corresponding global optimum solution is $\vec{x}_{opt} = (x_1, x_2, \ldots, x_d) = (0, 0, \ldots, 0)$.

The fourth function ($f_4(\vec{x})$) is the Rosenbrock function (Rosenbrock, 1960), which has a long and narrow parabolic flat valley, where the global minimum is located . $\vec{x}$ is defined in the range of $[-30, 30]$ and the global minimum value for $f_4(\vec{x})$ is 0 and the corresponding global optimum solution is $\vec{x}_{opt} = (x_1, x_2, \ldots, x_d) = (1, 1, \ldots, 1)$.

Since the BSO is a maximization algorithm and these four functions are meant to be minimized, a conversion is needed. Simply multiplying the fitness by -1 will not work, because the roulette wheel selection method accepts only non-negative fitness values. Therefore, the following transformation shown in Equation 3 was applied to it.

$$fit(\vec{x}) = \frac{k}{k + f(\vec{x})} \tag{3}$$

where $fit(\vec{x})$ is the corrected fitness to be maximized, $f(\vec{x})$ is the function raw value (which we want to minimize), and k is a given constant. In our experiments, functions f_1, f_2 and f_3 used $k = 1$, and f_4 used $k = 100$.

Should be noted that Equation 3 guarantees fitness values in the interval [0,1] for the tested functions, which is needed for the step sizing process. In fact, if the function to be minimized can produce negative fitness values, a more elaborate transformation will be needed. As mentioned before, the fitness values must always be in the [0,1] interval.
The BSO was compared to the PSO algorithm using cognitive and social constants $\phi_1 = \phi_2 = 1.8$, and the inertia weight was set to $\omega = 0.6$, as recommended in (Kennedy & Eberhart, 2001; Vesterstrom & Thomsen, 2004).
The number of variables (dimensions) for all functions in our experiments was set to 10, 30 and 50. In all experiments, the maximum number of function evaluations and swarm size were set to 500,000 and 500 particles, respectively. These two parameters were used both in the BSO and the PSO algorithm. For all models, the stop condition is reaching the maximum number of evaluations or a 10^{-5} error.
The number of iterations for each local search procedure in the BSO were set to $n_w = 10$ and $n_s = 100$, meaning that the strong local search is ten times heavier than the weak local search. The initial radius for each one was set to $r_{0w} = 0.1$ and $r_{0s} = 1.0$, and the sampling radius decrease rate of the LUS was set to $q = 0.6$.

4.1 Parameter tuning

We have also investigated the behavior of BSO to different settings of some control parameters. The objective is to suggest default values, whenever possible.
Parameters ρ and γ were fixed using the default values from GSO, that is, $\rho = 0.4$ and $\gamma = 0.6$. The other parameters were tested using a factorial experiment (Box et al., 2005) with the following values: $n = [50; 300; 500; 1000]$, $s_0 = [0.3; 1.0; 3.0]$, $lR = [0; 1; 5; 10; 50]$, $c_g = [0.01; 0.03; 0.1]$, $c_s = [1; 5; 10]$, and $eT = [100; 200]$. A total of 1080 possible combinations of these parameter values can be arranged. Experiments were done with all these combinations of parameters, for each function of each dimension. Each parameter set was tested 20 times, in independent runs, and the average best result found for each test was used later to define a default parameter set.
In almost all problems, the bigger the population size n , the better the result, with $n = 1000$ achieving the best solutions, in average. However, in f_2 and f_4 for $d = 30$ and $d = 50$, the best results were found using $n = 50$. This happened because the maximum number of function evaluations was reached. The BSO has a very slow convergence, and the number of iterations in which the limit of function evaluations was reached with large populations was not enough to converge. In fact, even with a small population, the convergence curve still showed some room to improvement, meaning that this limited number of evaluations was too small for these instances of the problems. Therefore, we set as the default value $n = 500$, since it lead to results almost as good as those found with $n = 1000$ in most problems, but had much better solutions than this value when more time was needed by the algorithm.
The parameter s_0 showed to be strongly dependent of the problem. For each instance of the problems, a different value of s_0 leaded to better solutions. This was expected, since this parameter has a direct influence in the navigation steps in the search space. In search spaces with different topologies, different values of s_0 will be needed. The range used in our experiments achieved good results for all functions. Therefore, s_0 should be usually set in the range $[0.3..3.0]$. In our experiments, we used $s_0 = 1.0$ as the default value.
The global attraction, controlled by the c_g parameter, lead to better results when set to the lower values. The only exception was in f_4. However, even in this function, small values also yielded good solutions. Consequently, we defined $c_g = 0.03$ as the default value for this

parameter, since a good performance in all instances of the problems was achieved with this value.

The stochastic step sizing, which controls the thoroughness of the search that each particle performs, showed better results when the exploitation was privileged. This means higher values of c_s. As explained in Section 3.2, higher values of c_s lead particles to explore the space with slow movements, even in regions with not so high fitness levels. This parameter leads to a trade-off between exploration and exploitation, and the user should tune it according to what it is wanted to emphasize. In almost all experiments, good results were found using $c_s = 5$, and this value is set as the default for this parameter.

Our experiments also showed that the eT parameter has low influence in the quality of the final results, but lower values had a slight advantage. Since good results were achieved in all functions using $eT = 100$, we consider this as the default value.

Usually, the strong local search procedure (see Section 3.5.2) showed better results with lower values. Lower values mean more local search, not the contrary, and so, the more effort spent in local search, the better the results. However, with $lR = 1$, too much function evaluations were spent in the local search, and the algorithm had few time to explore the search space. With small populations the effect was more pronounced, since the number of function evaluations used by SDPS was proportionally larger. When testing the functions with lower dimensions, this was not a problem, since the algorithm needed much less evaluations than the upper limit set. However, when using $d = 50$, higher values of lR lead to better solutions, but the quality dropped again when lR was set too high. Considering that $lR = 5$ leaded to good results for all problems, this is defined as the default.

Table 2 summarizes the default values for the control parameters of BSO. Notice that the maximum number of iterations $maxIte$, as well as the maximum number of evaluations cannot be considered parameters of the BSO. They only control the stop criterion, which can be different for each problem and application of the algorithm. One can set the stop criterion to a predefined error value, number of fitness evaluations or processing time spent, for instance.

Symbol	Name	Default Value
n	Population size	500
ρ	Luciferin decay constant	0.4
γ	Luciferin gain constant	0.6
s_0	Maximum step size	[0.3 .. 3.0]
eT	Number of stagnated iterations to enable explosion	100
lR	Period of local search procedures	5
c_g	Attraction to global best	0.03
c_s	Slowing constant	5

Table 2. Default values for the control parameters of BSO

5. Results and discussion

In this section we present results of our experiments and a comparison of performance between PSO and BSO, as well as a study of the convergence behavior of each algorithm.

5.1 Numerical results

The statistical results of 100 independent runs obtained by BSO and PSO with default parameters are shown in Tables 3 to 6. BSO was tested with and without the strong local

search (SDPS). In the tables, besides the full BSO algorithm, it is also shown wBSO, a version of BSO without the strong local search. Each column shows the average of the best values obtained in each run followed by the respective standard deviations, and the mean of the elapsed time in seconds by each algorithm for each function.
In Table 3, the results for function f_1 indicate that the BSO algorithm has an overhead significantly larger than PSO for small dimensions, but this overhead grows much slower in higher dimensions. With $d = 10$, PSO was almost 4 times faster than BSO. On the other hand, they took almost the same time to execute when $d = 50$. The quality of the solutions found by the BSO were also much better than the PSO. The reason for this, further explained in Section 5.2, is the slower convergence of BSO, avoiding getting trapped into local maxima.

	$d = 10$		$d = 30$		$d = 50$	
Model	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
BSO	0.00005 ± 0.00004	2.34	0.14368 ± 0.38712	2.84	0.32219 ± 0.80927	3.38
wBSO	0.00688 ± 0.00201	2.45	0.16968 ± 0.10514	3.04	0.88320 ± 0.59278	3.61
PSO	7.79839 ± 3.69998	0.61	27.8135 ± 7.41229	1.74	88.7282 ± 17.0144	3.04

Table 3. Statistical results obtained by all approaches - Function f_1

The results for function f_2, shown in Table 4, have the same time characteristics found in f_1. That is, large overhead for small dimensions, but equivalent performance for high dimensions. The quality of the results were close between PSO and BSO, with better solutions found by PSO than wBSO when $d = 30$ and $d = 50$.

	$d = 10$		$d = 30$		$d = 50$	
Model	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
BSO	0.03465 ± 0.02183	2.54	0.02628 ± 0.02542	3.15	0.02919 ± 0.01673	3.81
wBSO	0.08019 ± 0.02994	2.64	0.43223 ± 0.11680	3.39	0.74025 ± 0.09198	4.15
PSO	0.09003 ± 0.03284	0.67	0.23263 ± 0.09442	1.97	0.66629 ± 0.10810	3.28

Table 4. Statistical results obtained by all approaches - Function f_2

Concerning function f_3, Table 5 shows that BSO again outperformed PSO, and the difference between the time spent by each algorithm also decreased for higher dimensions.

	$d = 10$		$d = 30$		$d = 50$	
Model	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
BSO	0.07870 ± 0.02445	2.50	0.50525 ± 0.18516	3.39	1.39567 ± 0.55424	4.26
wBSO	0.09584 ± 0.02414	2.60	1.21714 ± 0.33930	3.52	3.36662 ± 1.10405	4.41
PSO	0.53988 ± 0.20318	0.62	5.60456 ± 0.90357	1.85	12.7432 ± 1.35718	3.10

Table 5. Statistical results obtained by all approaches - Function f_3

Results for function f_4, shown in Table 6, demonstrate that the time spent by PSO when $d = 50$ was higher than the time spent by the BSO. Overall, this indicates that BSO can be faster than PSO if the number of dimensions is high enough. Here, the quality of the solutions found degraded fastly for PSO, comparing with BSO, as the number of dimensions increased.

5.2 Convergence analysis

An important feature of the BSO algorithm is a very slow convergence, when compared, for instance, with PSO. To show that, we ran a set of tests without the strong local search (using

	$d = 10$		$d = 30$		$d = 50$	
Model	Quality	Time (s)	Quality	Time (s)	Quality	Time (s)
BSO	0.72827 ± 1.52126	2.16	27.0083 ± 1.75217	2.33	47.0415 ± 0.79140	2.48
wBSO	2.06372 ± 1.75578	2.23	33.0884 ± 10.4737	2.39	79.7785 ± 36.1792	2.56
PSO	0.92752 ± 2.07380	0.64	67.6840 ± 30.3727	1.86	290.274 ± 171.253	3.10

Table 6. Statistical results obtained by all approaches - Function f_4

wBSO), since the weak local search is considered the default movement for the best particle, and the PSO do not have a specific local search procedure in this implementation.
The results of these experiments are in Figures 1 to 4. In these figures, the fitness for both methods are normalized by the best fitness found by each one. This means that there is no information at all about the quality of the solutions found by each approach in these figures, only about the convergence. The quality of the solutions was previously discussed in Section 5.1. These figures shows that BSO usually has a very slow start, then accelerate its convergence speed, but still much slower than PSO. In f_1, the BSO converged around iteration 400, while PSO converged around iteration 100, see Figure 1. This slower convergence led to a much better final result, as shown in Table 3.

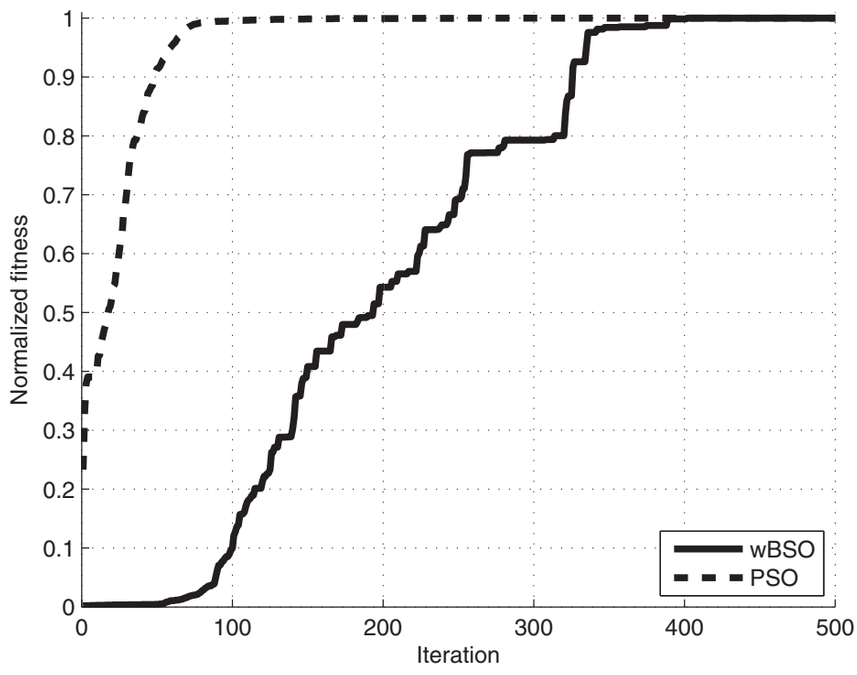


Fig. 1. Convergence curve for wBSO and PSO in f_1, d=50

The convergence curves for function f_2, shown in Figure 2, shows that BSO and PSO converged around iteration 700 and 200, respectively. However, the quality of the solution was not that good, as shown in Table 4.
Concerning function f_3, the curves shown in Figure 3 indicate that convergence ocurred when BSO reached approximately 500 iterations, and PSO found its final solution as soon as iteration 100, what lead to very bad results.

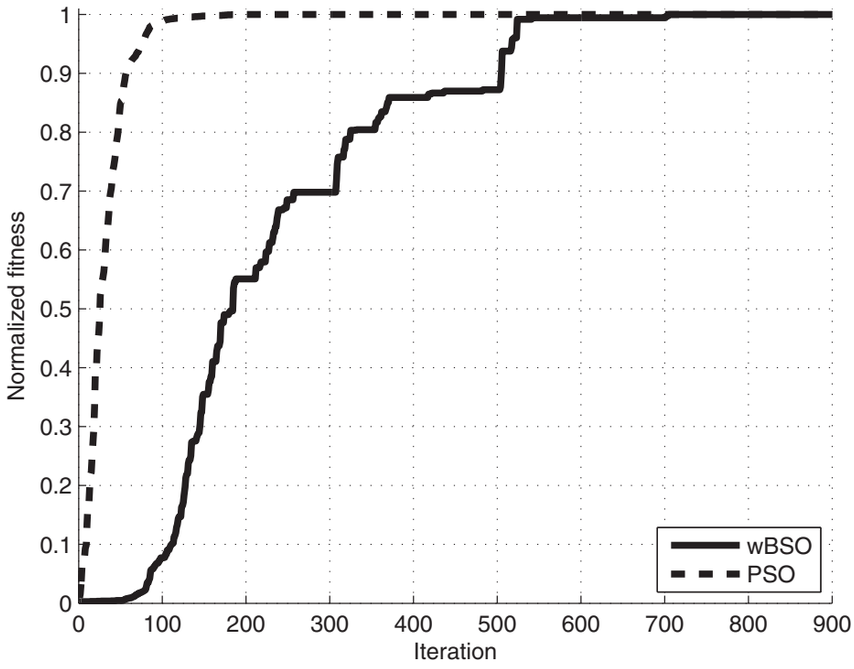


Fig. 2. Convergence curve for wBSO and PSO in f_2, d=50

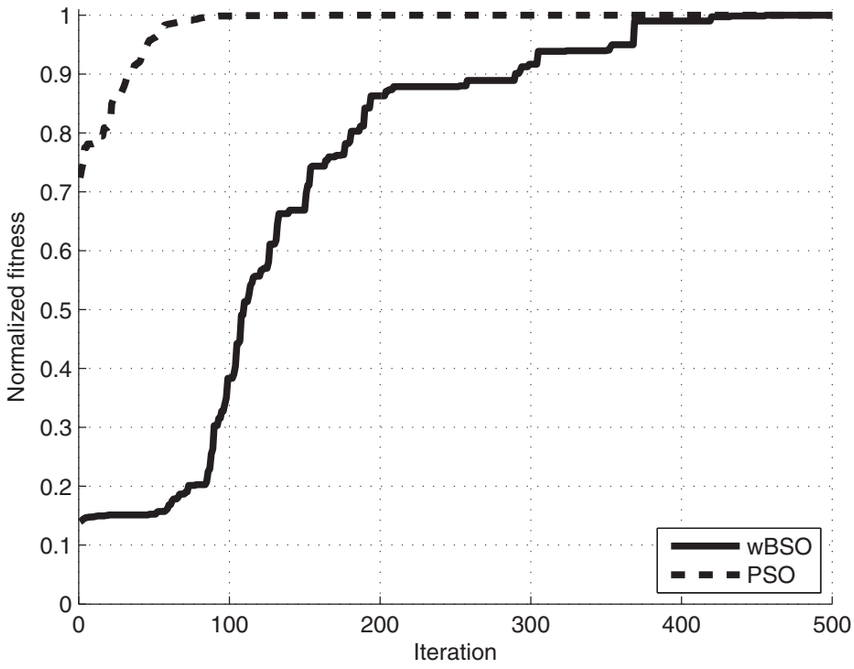


Fig. 3. Convergence curve for wBSO and PSO in f_3, d=50

For function f_4, Figure 4 shows that the BSO algorithm reached 900 iterations with presumably some potential to improve the solution, while the PSO stopped near 250 iterations.

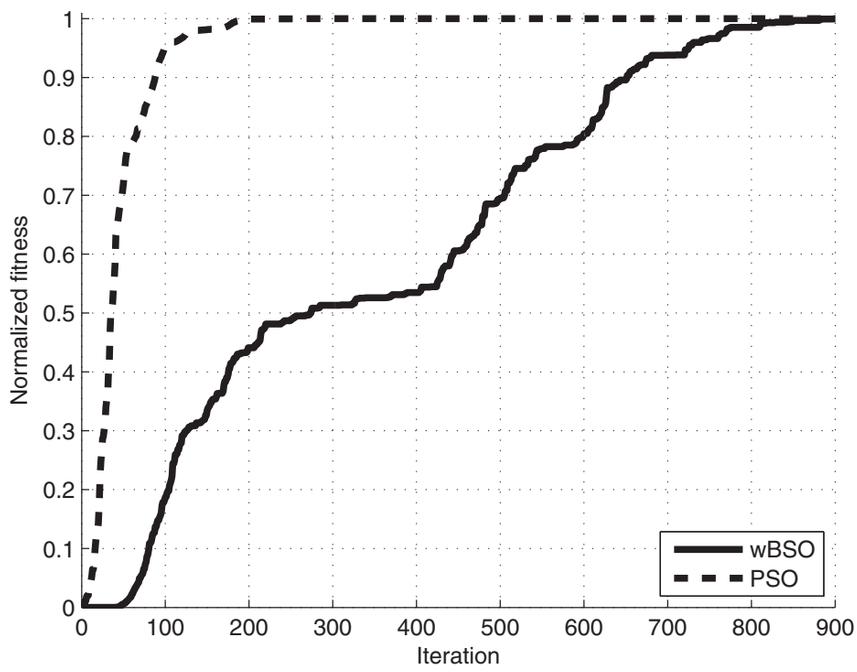


Fig. 4. Convergence curve for wBSO and PSO in f_4, d=50

In these four studies, using functions with 50 dimensions, BSO converged between 4 to 5 times slower than PSO. As consequence, this lead to very good solutions, since fast convergence usually makes the algorithm to get trapped into local maxima. The results were similar for the other dimensions studied (not shown here), always with a smooth convergence in the BSO algorithm.

6. Conclusions and future work

In this work we proposed a swarm-based algorithm for global optimization named Bioluminescent Swarm Optimization (BSO) algorithm. The algorithm is based both on research on the behavior of real *Lampyridae* family of insects and on Swarm Intelligence concepts and principles.

The BSO algorithm has a very good exploration capability, avoiding getting trapped into local maxima, while its local search procedures improved the otherwise somewhat weak exploitation. However, the introduction of local search did not remove the slow convergence characteristic, leading to high-quality final results with a smooth convergence.

The experiments concerning the BSO robustness suggested that the algorithm has low sensitivity to parameter tuning. Hence, in most cases, the default parameters proposed here should lead to good results just by tuning the s_0 parameter and the population size n.

We have compared the performance of BSO and one of the most widely used swarm intelligence method (PSO) in four benchmark functions, using three different number of dimensions for each one. Results obtained by the BSO algorithm were much better than those

obtained by the other approach, indicating that the proposed algorithm is an interesting and promising strategy for global optimization of complex problems.
Future work will address the selection system, self-tuning of the parameters, and multi-objective optimization. We will also apply the BSO algorithm to other benchmark functions, and real-world problems. We will investigate other local search methods to be used as the strong local search procedure, since this approach proved to be very promising. In the same way, hybridizing the proposed algorithm with other evolutionary computation methods may lead to improved performance.

7. Acknowledgements

This work was partially supported by the Brazilian National Research Council (CNPq) under research grant no. 309262/2007-0. Authors would like to thank UDESC (Santa Catarina State University) and the FUMDES program for the financial support.

8. References

Benítez, C. & Lopes, H. (2010). A master-slave parallel genetic algorithm for protein structure prediction using the 3D-HP side-chain model, *Journal of the Brazilian of Computer Society* 16(1): 69–78.

Box, G., Hunter, W. & Hunter, J. (2005). *Statistics for Experimenters: Design, Innovation, and Discovery*, 2nd edn, Wiley, New York.

Cho, H., Olivera, F. & Guikema, S. (2008). A derivation of the number of minima of the Griewank function, *Applied Mathematics and Computation* 204(2): 694–701.

Clerc, M. (2006). *Particle Swarm Optimization*, Wiley-ISTE Press, London.

Digalakis, J. G. & Margaritis, K. G. (2002). An experimental study of benchmarking functions for evolutionary algorithms, *International Journal of Computer Mathematics* 79(4): 403–416.

Dorigo, M. & Stützle, T. (2004). *Ant Colony Optimization*, MIT Press, Cambridge.

Feng, X., Lau, F. C. M. & Gao, D. (2009). A new bio-inspired approach to the traveling salesman problem, *in* X. Feng, F. C. Lau & D. Gao (eds), *Complex Sciences*, Vol. 5 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Berlin/Heidelberg, pp. 1310–1321.

Floudas, C. A. & Pardalos, P. M. (1990). *A collection of test problems for constrained global optimization problems*, Vol. 455 of *Lecture Notes in Computer Science*, Springer.

Fraga, H. (2008). Firefly luminescence: A historical perspective and recent developments, *Journal of Photochemical & Photobiological Sciences* 7: 146–158.

Geem, Z., Kim, J. & Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search, *Simulation* 76(2): 60–68.

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning*, Addison Wesley, Reading.

Griewank, A. (1981). Generalized descent for global optimization, *Journal of Optimization Theory and Applications* 34(1): 11–39.

Havens, T., Spain, C., Salmon, N. & Keller, J. (2008). Roach infestation optimization, *IEEE Swarm Intelligence Symposium* pp. 1–7.

Hembecker, F., Godoy Jr., W. & Lopes, H. (2007). Particle swarm optimization for the multidimensional knapsack problem, *Lecture Notes in Computer Science* 4331: 358–365.

Kalegari, D. & Lopes, H. (2010). A differential evolution approach for protein structure optimization in a 2-D off-lattice protein model, *International Journal of Bio-Inspired Computation* 2(3/4): 242–250.

Karaboga, D. & Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence, *Artificial Intelligence Review* 31(1/4): 61–85.

Kennedy, J. & Eberhart, R. (2001). *Swarm Intelligence*, Morgan Kaufmann, San Francisco.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge.

Krishnanand, K. & Ghose, D. (2005). Detection of multiple source locations using a glowworm metaphor with applications to collective robotics, *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 84–91.

Krishnanand, K. & Ghose, D. (2008). Glowworm swarm optimization algorithm for hazard sensing in ubiquitous environments using heterogeneous agent swarms, *in* B. Prasad (ed.), *Soft Computing Applications in Industry*, Vol. 226 of *Studies in Fuzziness and Soft Computing*, Springer-Verlag, Heidelberg, pp. 165–187.

Krishnanand, K. & Ghose, D. (2009). Glowworm swarm optimization for simulataneous capture of multiple local optima of multimodal functions, *Swarm Intelligence* 3(2): 87–124.

Mühlenbein, H., Schomisch, D. & Born, J. (1991). The parallel genetic algorithm as function optimizer, *Parallel Computing* 17(6-7): 619–632.

Monismith, D. & Mayfield, B. (2008). Slime mold as a model for numerical optimization, *IEEE Swarm Intelligence Symposium*, pp. 1–8.

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control, *IEEE Control Systems Magazine* 22(3): 52–67.

Pedersen, M. E. H. (2010). *Tuning & Simplifying Heuristical Optimization*, Phd thesis, School of Engineering Sciences, University of Southampton, UK.

Poli, R., Kennedy, J. & Blackwell, T. (2007). Particle swarm optimization: an overview, *Swarm Intelligence* 1(1): 33–57.

Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function, *The Computer Journal* 3: 175–184.

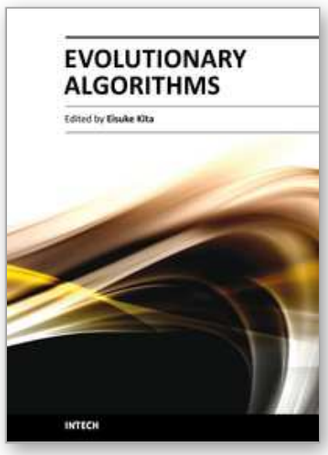
Shimomura, O. (2006). *Bioluminescense: Chemical Principles and Methods*, World Scientific Publishing, Singapore.

Storn, R. & Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* 11: 341–359.

Vesterstrom, J. & Thomsen, R. (2004). A comparative study of differential evolution particle swarm optimization and evolutionary algorithms on numerical benchmark problems, *IEEE Congress on Evolutionary Computation* 3: 1980–1987.

Yang, X. (2010). A new metaheuristic bat-inspired algorithm, *Nature Inspired Cooperative Strategies for Optimization*, Vol. 284 of *Studies in Computational Intelligence*, Heidelberg, pp. 65–74.

Yang, X. S. (2009). Firefly algorithms for multimodal optimization, *Lecture Notes in Computer Sciences* 5792: 169–178.



Evolutionary Algorithms
Edited by Prof. Eisuke Kita

ISBN 978-953-307-171-8
Hard cover, 584 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

Evolutionary algorithms are successively applied to wide optimization problems in the engineering, marketing, operations research, and social science, such as include scheduling, genetics, material selection, structural design and so on. Apart from mathematical optimization problems, evolutionary algorithms have also been used as an experimental framework within biological evolution and natural selection in the field of artificial life.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen