Proceedings of the IASTED International Conference
Artificial Intelligence and Applications (AIA 2014)
February 17 - 19, 2014 Innsbruck, Austria

TOWARDS THE SELF-ADAPTATION OF THE BAT ALGORITHM

Iztok Fister Jr.[a], Simon Fong[b], Janez Brest[a], Iztok Fister[a]
[a] University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenia
Email: iztok.fister2@uni-mb.si, janez.brest@uni-mb.si, iztok.fister@uni-mb.si
[b] University of Macau, Faculty of Science and Technology, Av. Padre Tomas Pereira, Taipa, Macau SAR
Email: ccfong@umac.mo

ABSTRACT
Bat algorithm is one of the younger members of swarm intelligence algorithms that was introduced by Yang in 2010. So far, several variants based on this algorithm have been proposed for coping with the continuous and discrete problems. This paper introduces a novel self-adaptive bat algorithm (SABA) that borrows the self-adapting mechanism from self-adapted differential evolution known under the name jDE. This algorithm was tested on ten benchmark functions taken from publications. The obtained results were very promising and showed that this algorithm might be very suitable for use, especially, in the continuous optimization.

KEY WORDS
bat algorithm, swarm intelligence, optimization, self-adaptation

1 Introduction

Nowadays, optimization has become more and more important in domains, like finances, industries, sport, pharmacy, etc. The task of the optimization is to find the minimum or maximum of objective function relating to a problem that needs to be solved. Typically, the minimal costs of production are searched for in many industries. For example, the problem of how to finish a specific work order with minimal material usage has arisen in the clothing industry [1]. However, the clothing industry is not alone example.

The similar optimization problems have been arisen in many industries that are efficiently solved using the well-known algorithms, like:

- artificial bee colony [2],
- bat algorithm [3],
- differential algorithm [4],
- firefly algorithm [5],
- particle swarm optimization [6], and
- many more [7].

The mentioned algorithms are generally inspired by nature. In these algorithms, developers wish to mimics the natural behavior of different biological systems in order to solve particular problems using the digital computers. Adaptation is one of the more important characteristic for surviving the individuals in biological systems that are subject to ruthless changing environment [8]. This mechanism enables the individuals in biological systems to adapt themselves to the changing conditions within their environments. The adaptation is not just a unique characteristic of the biological systems, because this mechanism has also been incorporated into artificial systems and optimization algorithms.

The optimization algorithms are controlled by algorithm parameters that can be changed deterministically, adaptively, and self-adaptively [9]. Deterministic parameters are altered by using some deterministic rule (e.g., Rechenberg's 1/5 success rule [10]). In contrast, adaptive controlled parameters are subject to feedback from the search process that serves as an input to a mechanism used that determines the direction and magnitude of the change [9]. Finally, the self-adaptive controlled parameters are encoded into representation of solution and undergo acting the variation operators, like crossover and mutation in evolutionary algorithms [11].

This paper proposes a self-adaptive bat algorithm (SABA) for continuous optimization. However, this self-adaptive variant of a bat algorithm (BA) can also be later applied to real-world problems. An origin of the BA algorithm goes to the year 2010, when Xin-She Yang [12, 13, 14] created a new optimization algorithm inspired by the behavior of micro-bats that use a special mechanism called echolocation. Echolocation is used by bats for orientation and prey seeking. The original BA was applied to various benchmark functions, in which it achieved the good results. The convergence rate was improved in study [3], where the authors hybridized the original BA with differential evolution strategies (HBA). On the other hand, Wang and Guo [15] successfully hybridized the BA with a harmony search (HS/BA) [16].

The original bat algorithm employs two strategy parameters: the pulse rate and the loudness. The former regulates an improving of the best solution, while the later influences an acceptance of the best solution. Both the mentioned parameters are fixed during the execution of the original bat algorithm. In SABA, these parameters are self-adapted. The aim of this self-adaptation is twofold. On the

DOI: 10.2316/P.2014.816-011

one hand, it is very difficult to guess the valid value of the parameter. On the other hand, this value depends on the phase in which the search process is. This means, the parameter setting at the beginning of the search process can be changed, when this process becomes matured. Therefore, the self-adaptation of these parameters is applied in SABA.

The proposed SABA algorithm were applied to an optimization of a benchmark function suite consisting of 10 well-known functions from the publications. The obtained results using SABA improved the results achieved by the original BA. Additionally, the proposed algorithm was compared also with the other well-known algorithms, like firefly (FA) [5], differential evolution (DE) [17], and artificial bee colony (ABC) [18]. The results from this comparison showed that the results of the SABA are comparable with the results of the other up-to-date algorithms.

The structure of this paper is as follows. Section 2 presents the original bat algorithms. The self-adaptive bat algorithm is described in Section 3. Section 4 illustrates the experiments and results detailed. The paper concludes by summarizing the performed work and outlines further directions for development.

2 The original bat algorithm

The original bat algorithm was developed in 2010 by Xin-She Yang [12, 13, 14]. The inspiration for his work came from behavior of micro-bats and especially, their special mechanism named echolocation. Echolocation is a mechanism that bats use for orientation and finding the preys. Bats are not the only creatures using such a mechanism. For instance, dolphins use this mechanism for finding prey in the seas [19].

Nowadays, BA and its variants are applied for solving many optimization and classification problems, as well as several engineering problems in practice. In detail, the taxonomy of the developed BA applications is represented in [19], where the applications of BA algorithms have been divided into the following classes of optimization problems: continuous, constrained, and multi-objective. In addition, they were also used for classification problems, like clustering, neural networks, and feature selection. Finally, BAs were used in many branches of engineering, e.g., image processing, industrial design, scheduling, electronics, spam filtering, and even in sport.

The original bat algorithm is population based, where each individual represents the candidate solution. The candidate solutions are represented as vectors $\mathbf{x_i} = (x_{i1}, \ldots, x_{iD})^T$ for $i = 1 \ldots Np$ with real-valued elements x_{ij}, where each elements can capture values from interval $x_{ij} \in [x_{lb}, x_{ub}]$. Thus, x_{lb} and x_{ub} denote the corresponding lower and upper bound, while the population size is determined by Np parameter.

The pseudo-code of the original BA algorithm is illustrated in Algorithm 1, where bats' behavior is captured within the fitness function of the problem to be solved. The

Algorithm 1 Pseudo-code of the original bat algorithm

Input: Bat population $\mathbf{x_i} = (x_{i1}, \ldots, x_{iD})^T$ for $i = 1 \ldots Np, MAX\_FE$.
Output: The best solution $\mathbf{x}_{best}$ and its corresponding value $f_{min} = \min(f(\mathbf{x}))$.

```
1:  init_bat();
2:  eval = evaluate_the_new_population;
3:  f_min = find_best_solution(x_best); {initialization}
4:  while termination_condition_not_meet do
5:    for i = 1 to Np do
6:      y = generate_new_solution(x_i);
7:      if rand(0,1) > r_i then
8:        y = improve_the_best_solution(x_best)
9:      end if{ local_search_step }
10:     f_new = evaluate_new_solution(y);
11:     eval = eval + 1;
12:     if f_new ≤ f_i and N(0,1) < A_i then
13:       x_i = y; f_i = f_new;
14:     end if{ save_best_solution_conditionaly }
15:     f_min=find_best_solution(x_best);
16:   end for
17: end while
```

operation of the original BA algorithm presented in Algorithm 1 can be described as follows. In general, the algorithm consists of the following components [19]:

- *initialization* (lines 1-3): initializing the algorithm parameters, generating the initial population, evaluating it, and searching for the best solution $\mathbf{x}_{best}$ in the initial population,
- *generate_new_solution* (line 6): moving the virtual bats in the search space according to physical rules of bat echolocation,
- *local_search_step* (lines 7-9): improving the best solution using random walk direct exploitation (RWDE) heuristic [20],
- *evaluate_new_solution* (line 10): evaluating the new solution,
- *save_best_solution_conditionaly* (lines 12-14): saving the new best solution under some probability A_i similar to simulated annealing [21],
- *find_best_solution* (line 15): searching for the current best solution.

Note that these components are denoted in the algorithm either as function names, when the function call is performed in one line or as a component name designated by a comment between two curly brackets, when it comprises more lines. Initialization of the bat population consist of tree steps. At first, the population is initialized randomly (in function *init_bat*). Then, solutions are evaluated (in function *evaluate_the_new_population*)

and finally, the current best solution is found (in function *find_best_solution*). Generating the new solutions is performed according to the following equations:

$$\begin{aligned} Q_i^{(t)} &= Q_{min} + (Q_{max} - Q_{min})N(0,1), \\ \mathbf{v}_i^{(t+1)} &= \mathbf{v}_i^t + (\mathbf{x}_i^t - \mathbf{best})Q_i^{(t)}, \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}, \end{aligned} \tag{1}$$

where $N(0,1)$ is a random number generated from a Gaussian distribution with zero mean and a standard deviation of one. A RWDE heuristic [20] implemented in the function *improve_the_best_solution* modifies the current best solution according to the equation:

$$\mathbf{x^{(t)}} = \mathbf{best} + \epsilon A_i^{(t)} N(0,1), \tag{2}$$

where $N(0,1)$ denotes the random number generated from a Gaussian distribution with zero mean and a standard deviation of one, ϵ being the scaling factor, and $A_i^{(t)}$ the loudness. A local search is launched with the probability of pulse rate r_i. As already stated, the probability of accepting the new best solution in the component *save_the_best_solution_conditionaly* depends on loudness A_i. Actually, the original BA algorithm is controlled by two algorithm parameters: the pulse rate r_i and the loudness A_i. Typically, the rate of pulse emission r_i increases and the loudness A_i decreases when the population draws nearer to the local optimum. Both characteristics imitate natural bats, where the rate of pulse emission increases and the loudness decreases when a bat finds a prey. Mathematically, these characteristics are captured using the following equations:

$$A_i^{(t+1)} = \alpha A_i^{(t)}, \quad r_i^{(t)} = r_i^{(0)}[1 - \exp(-\gamma\epsilon)], \tag{3}$$

where α and γ are constants. Actually, the α parameter controls the convergence rate of the bat algorithm and therefore, plays a similar role as the cooling factor in the simulated annealing algorithm.

In summary, the origin of BA can be found in an PSO algorithm [6] hybridized with RWDE and simulated annealing heuristics. The former represents the local search that directs the bat search process towards improving the best solution, while the latter takes care of the population diversity. In other words, the local search can be connected with exploitation, while simulated annealing using the exploration component of the bat search process. The exploitation is controlled by the parameter r and exploration by the parameter A. As a result, the BA algorithm tries to explicitly control the exploration and exploitation components within its search process.

3 The self-adaptive bat algorithm

Self-adaptation of control parameters means that the algorithms control parameters (also strategy parameters) are encoded into representation of a candidate solution and undergo acting the variation operators. In our case, a developed self-adaptive bat algorithm (SABA) considers the self-adaptation of the control parameters, e.g., the loudness $A^{(t)}$, and the pulse rate $r^{(t)}$.

In line with this, the existing representation of candidate solutions in the original bat algorithm consisting of problem variables $(x_{i1}^{(t)}, \ldots, x_{iD}^{(t)})^T$ is widened with the control parameters $A^{(t)}$ and $r^{(t)}$ to $\mathbf{x}_i^{(t)} = (x_{i1}^{(t)}, \ldots, x_{iD}^{(t)}, A^{(t)}, r^{(t)})^T$ for $i = 1 \ldots Np$, where Np denotes the population size. Additionally, these control parameters are modified according to the following equations:

$$A^{(t+1)} = \begin{cases} A_{lb}^{(t)} + rnd_0(A_{ub}^{(t)} - A_{lb}^{(t)}) & \text{if } rnd_1 < \tau_1, \\ A^{(t)} & \text{otherwise}, \end{cases} \tag{4}$$

and

$$r^{(t+1)} = \begin{cases} r_{lb}^{(t)} + rnd_2(r_{ub}^{(t)} - r_{lb}^{(t)}) & \text{if } rnd_3 < \tau_2, \\ r^{(t)} & \text{otherwise}. \end{cases} \tag{5}$$

Note that the parameters τ_0 and τ_1 denotes the learning rates that were set, as $\tau_0 = \tau_1 = 0.1$, while rnd_i for $i = 1 \ldots 4$ designate the random generated value from interval $[0, 1]$. Thus, the values of control parameters are limited according to intervals represented in Table 1.

Table 1. Boundary values of parameters

Parameter	In	Upper and lower bound
$A^{(t)}$	$\in$	$[0.001, 0.1]$
$r^{(t)}$	$\in$	$[0.9, 1.0]$

The self-adapting part of the SABA algorithm is performed in the *generate_the_new_solution* function (line 6 in Algorithm 1). It is notable that the control parameters are modified in this function according to the learning rate τ_0 and τ_1. In case when $\tau_0 = \tau_1 = 0.1$, the control parameters of each 10th candidate solution are modified, in average. The modified control parameters have an impact on the application of the local search (component *local_search_step* lines 7-9 in Algorithm 1) as well as on the probability of saving the best solution (component *save_best_solution_conditionaly* lines 12-14 in Algorithm 1). The problem variables and the corresponding control parameters are saved by this component when it comes to saving.

This self-adapting procedure was inspired by Brest et al. [22] that proposed the self-adaptive version of DE, better known as jDE. This self-adaptive algorithm improves the results of the original DE significantly by continuous optimization.

4 Experiments and results

Two goals were aimed our experimental work, as follows: to show that SABA can improved the results of the original bat algorithm (BA), and to show that the results of SABA are comparable to the results of other well known algorithms, like firefly (FA) [5], differential evolution (DE) [17], and artificial bee colony (ABC) [18]. In line with this, the self-adaptive bat algorithm (SABA) was developed and applied to well known benchmark function suite taken from publications.

Function optimization was selected as a test bed problem for this study. The function optimization is included into a class of continuous optimization problems that can formally be defined as follows. Let $f(\mathbf{s})$ be an objective function, where $\mathbf{x} = (x_1, \ldots, x_D)$ denotes a vector of D design variables from a decision space $x \in S$. These design variables $x_j \in \{lb_j, ub_j\}$ are captured from an interval $[lb_j, ub_j]$, where $lb_j \in \mathbb{R}$ and upper bounds $ub_j \in \mathbb{R}$ are their lower and upper bounds, respectively. Then, the task of function optimization is to find the minimum of this objective function.

In the rest of the paper, the function in benchmark suite are presented, then some words are intended to the experimental setup, and at the end of this section, the results obtained by the experiments are discussed in detail.

4.1 Benchmark suite

The benchmark suite was composed of ten well-known functions selected from various publications. The definitions the benchmark functions are summarized in Table 2 that consists of four columns denoted: the function tag f, the *function name*, the function *definition*, and the parameter *domain*. Reader is invited to check a deep details about test functions in the state-of-the art reviews [23, 24, 25].

Each function from the table is tagged with its sequence number from f_1 to f_{10}. Parameter domains limit the values of parameters into interval between their lower and upper bounds. As matter of fact, these determine the size of the search space. In order to make the problems more heavier to solve, the parameter domains were selected wider that those prescribed in the standard publications. Additionally, the problem becomes also heavier to solve when the dimensionality of the benchmark functions are increased. As a result, benchmark functions of more dimensions need to be optimized in the experimental work.

Properties of the benchmark functions can be seen in Table 3 that also consists of four columns: the function tag f, the optimal solution x^*, the value of the optimal solution x^*, and the function *characteristics*. One of the more important characteristics of the function is the number of local and global optima. According to this characteristic the functions are divided either into uni-modal or multi-modal. The former type of functions has only one global optimum, while the later is able to have more local and global optima trowed across the whole search space.

[1] Valid for 2-dimensional parameter space.

Table 3. Properties of benchmark functions

f	f^*	x^*	Characteristics
f_1	0.0000	$(0, 0, \ldots, 0)$	Highly multi-modal
f_2	0.0000	$(0, 0, \ldots, 0)$	Highly multi-modal
f_3	0.0000	$(1, 1, \ldots, 1)$	Several local optima
f_4	0.0000	$(0, 0, \ldots, 0)$	Highly multi-modal
f_5	0.0000	$(0, 0, \ldots, 0)$	Highly multi-modal
f_6	0.0000	$(0, 0, \ldots, 0)$	Uni-modal, convex
f_7	-1.0000	$(\pi, \pi, \ldots, \pi)$	Several local optima
f_8	-1.8013[1]	$(2.20319, 1.57049)$[1]	Several local optima
f_9	0.0000	$(0, 0, \ldots, 0)$	Several local optima
f_{10}	0.0000	$(0, 0, \ldots, 0)$	Uni-modal

4.2 Experimental setup

In this experimental study, we compared the results of the following algorithms: BA, SABA, FA, DE, and ABC. During the tests, the BA parameters were set as follows: the loudness $A_0 = 0.5$, the pulse rate $r_0 = 0.5$, minimum frequency $Q_{max} = 0.0$, and maximum frequency $Q_{max} = 2.0$. The same initial value for r_0 and A_0 were also applied by SABA, while the frequency was captured from the same interval $Q \in [0.0, 2.0]$ as by the original bat algorithm. FA run with the following set of parameters: $\alpha = 0.1$, $\beta = 0.2$, and $\gamma = 0.9$, whilst DE was configured as follows: the amplification factor of the difference vector $F = 0.5$, and the crossover control parameter $CR = 0.9$. In the ABC algorithm, onlooker bees represented 50% of the whole colony, whilst the another 50% of the colony was reserved for the employed bees. On the other hand, the scout bee was generated when its value was not improved in 100 generations. In other words, the parameter $limits$ was set to value 100.

Each algorithm in tests were run with population size 100. Each algorithm was launched 25 times. The obtained results of these algorithms were aggregated according to their *Best*, the *Worst*, the *Mean*, the *StDev*, and the *Median* values reached during 25 runs.

4.3 The results

The intention of our experimental work was to show that the self-adaptive bat algorithm (SABA) can improve the results the original bat (BA), when applied to a test suite of ten benchmark functions. On the other hand, we want to show that the obtained results are comparable with the results of the other algorithms in tests, like FA, DE and ABC.

In line with this, each algorithm solved the functions of three different dimensions, i.e., $D = 10$, $D = 30$, and $D = 50$. As a termination condition, the maximum number of fitness function evaluations $MAX\_FE = 1000.D$

Table 2. Definitions of benchmark functions

f	Function name	Definition	Domain
f_1	Griewangk's function	$F(\mathbf{x}) = -\prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + \sum_{i=1}^{n} \frac{x_i^2}{4000} + 1$	$[-600, 600]$
f_2	Rastrigin's function	$F(\mathbf{x}) = n * 10 + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$	$[-15, 15]$
f_3	Rosenbrock's function	$F(\mathbf{x}) = \sum_{i=1}^{n-1} 100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	$[-15, 15]$
f_4	Ackley's function	$F(\mathbf{x}) = \sum_{i=1}^{n-1} \left(20 + e^{-20} e^{-0.2\sqrt{0.5(x_{i+1}^2 + x_i^2)}} - e^{0.5(\cos(2\pi x_{i+1}) + \cos(2\pi x_i))}\right)$	$[-32.768, 32.768]$
f_5	Schwefel's function	$f_5(\mathbf{x}) == 418.9829 * D - \sum_{i=1}^{D} s_i \sin(\sqrt{\lvert s_i \rvert})$	$[-500, 500]$
f_6	De Jong's sphere function	$f_6(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$[-600, 600]$
f_7	Easom's function	$f_7(\mathbf{x}) = -(-1)^D (\prod_{i=1}^{D} \cos^2(x_i)) \exp[-\sum_{i=1}^{D}(x_i - \pi)^2]$	$[-2\pi, 2\pi]$
f_8	Michalewicz's function	$f_8(\mathbf{x}) = -\sum_{i=1}^{D} \sin(x_i)[\sin(\frac{ix_i^2}{\pi})]^{2 \cdot 10}$	$[0, \pi]$
f_9	Xin-She Yang's function	$f_9(\mathbf{x}) = (\sum_{i=1}^{D} \lvert x_i \rvert) \exp[-\sum_{i=1}^{D} \sin(x_i^2)]$	$[-2\pi, 2\pi]$
f_{10}	Zakharov's function	$f_{10}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 + (\frac{1}{2}\sum_{i=1}^{D} ix_i)^2 + (\frac{1}{2}\sum_{i=1}^{D} ix_i)^4$	$[-5, 10]$

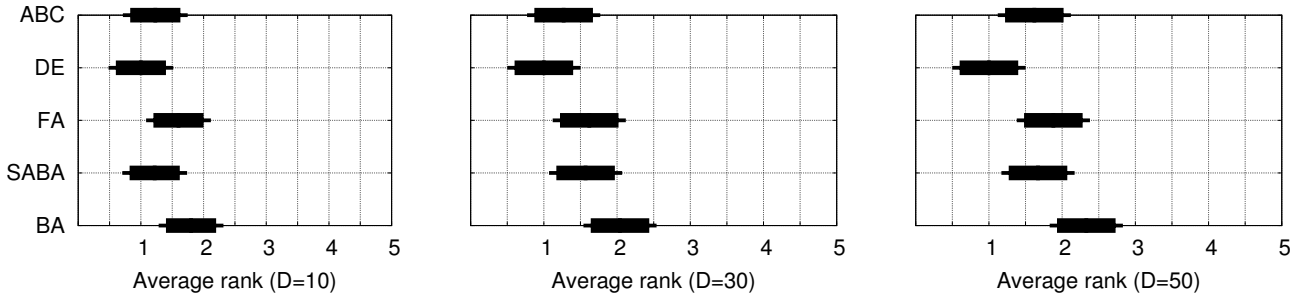

Figure 1. Friedman non-parametric tests

was used. In line with this, the 10-dimensional functions needed 10,000, the 30-dimensional 30,000, and the 50-dimensional 50,000 fitness function evaluations, in order to make the comparison fair. This means that the more dimensional functions had more evaluations to obtain the final solutions.

The results of optimizing the benchmark functions are presented in Table 4. Although the results were also obtained on the functions of all three dimensions, only the results of those functions with a dimension $D = 30$ are presented because of the limitation of paper length. The best results are displayed as bold. As can be seen from Table 4, ABC outperformed the results of all other algorithms four times (f_2, f_4, f_8, and f_9), FA three times (f_1, f_3, and f_7), while the SABA and DE were the best one time (f_6, and f_{10}) according to the mean values.

In order to evaluate the quality of results statistically, Friedman tests [26, 27] were conducted that compare the average ranks of the compared algorithms. Thus, a null-hypothesis is placed that states: two algorithms are equivalent and therefore, their ranks should be equal. When the null-hypothesis is rejected, the Bonferroni-Dunn test [28] is performed. In this test, the critical difference between the average ranks of those two algorithms is calculated. If the statistical difference is higher than the critical difference, the algorithms are significantly different.

Three Friedman tests were performed regarding data obtained by optimizing ten functions of three different dimensions according to five measures. As a result, each algorithm during the tests (also the classifier) was compared with regard to the 10 functions x 5 measures this means, 50 different variables. Tests were conducted at the significance level 0.05. The results of the Friedman non-parametric test can be seen in Fig. 1 that is divided into three diagrams. Each diagram shows the ranks and confidence intervals (critical differences) for the algorithms under consideration with regard to the dimensions of functions. Note that the significant difference between two algorithms is observed if their confidence intervals denoted as thickened lines in Fig. 1 do not overlap.

The first diagram in Fig. 1 shows that there are not any algorithm, which significantly outperforms the results of the other algorithms, according to dimension $D = 10$. From amongst all the five four algorithms, the DE was slightly better than the SABA and ABC, and substantially better than the FA and BA. The situation remained similar even when the results were compared regarding the dimensions $D = 30$ and $D = 50$. In the former case, the DE and ABC significantly improved the results of the BA, while in the latter case, the DE was significantly better than the FA and BA.

In summary, the SABA outperformed the results of

Table 4. Obtained results of algorithms (D=30)

Function	Measure	BA	SABA	FA	DE	ABC
f_1	Mean	1.16E+000	1.05E+000	**6.65E-001**	1.05E+000	1.09E+000
	Stdev	3.55E-002	3.45E-002	1.50E-001	2.22E-002	1.23E-001
f_2	Mean	9.28E+002	6.46E+002	2.44E+002	2.28E+002	**7.33E+001**
	Stdev	1.73E+002	1.30E+002	1.79E+001	1.33E+001	2.24E+001
f_3	Mean	2.84E+006	4.67E+005	**1.12E+002**	4.57E+002	5.18E+002
	Stdev	9.58E+005	3.87E+005	3.11E+001	2.27E+002	4.72E+002
f_4	Mean	2.00E+001	2.00E+001	2.11E+001	**1.77E+000**	7.17E+000
	Stdev	3.70E-006	5.59E-006	5.79E-002	3.17E-001	1.03E+000
f_5	Mean	8.14E+003	8.28E+003	6.78E+003	7.57E+003	**2.64E+003**
	Stdev	5.49E+002	6.92E+002	5.51E+002	4.40E+002	3.30E+002
f_6	Mean	5.87E-002	**1.51E-005**	5.19E+000	1.77E+002	1.63E+002
	Stdev	1.18E-001	1.97E-006	1.72E+000	7.12E+001	1.96E+002
f_7	Mean	0.00E+000	0.00E+000	**-3.81E-030**	0.00E+000	0.00E+000
	Stdev	0.00E+000	0.00E+000	1.09E-030	0.00E+000	8.79E-136
f_8	Mean	-8.62E+000	-8.13E+000	-5.15E+000	-1.07E+001	**-2.30E+001**
	Stdev	1.34E+000	1.44E+000	1.47E+000	6.70E-001	6.98E-001
f_9	Mean	5.56E-003	3.64E-003	1.70E-004	2.46E-011	**1.10E-011**
	Stdev	6.33E-002	7.91E-002	1.78E-004	1.20E-012	1.91E-012
f_{10}	Mean	2.76E+002	1.91E+002	1.32E+004	**3.78E+001**	2.53E+002
	Stdev	1.03E+002	1.15E+002	4.10E+001	8.74E+000	3.15E+001

the original BA substantially as can be seen in Fig. 1. Interestingly, the critical difference between both algorithms does not decrease when the dimension of the functions is increased (e.g., $D = 50$). Therefore, we could reasonable assume that self-adaptation could improve the results of the SABA by optimization of higher-dimensional functions. However, this assumption must be confirmed in the future. Additionally, the SABA outperformed also the results of the FA algorithm, while the results were comparable with the results of the DE and ABC.

5 Conclusion

In this paper, we proposed a self-adaptive variant of the bat algorithm named SABA. In this algorithm, control parameters are adapted in the similar way as by the self-adaptive DE algorithm known under the name jDE [22]. The experimental results showed an improvement of the performance of the proposed algorithm that encourages us to continue with the started experiments. We hope that this algorithm may be suitable for solving problems of higher dimensions. In the future, we would also wish to apply this self-adaptive method to other heuristic algorithms, e.g. cuckoo search.

Acknowledgement

The authors are thankful for the financial support from the research grant of Grant no. MYRG152(Y3-L2)-FST11-ZY, offered by the University of Macau, RDAO.

References

[1] I. Fister, M. Mernik, and B. Filipič, "A hybrid self-adaptive evolutionary algorithm for marker optimization in the clothing industry," *Applied Soft Computing*, vol. 10, no. 2, pp. 409–422, 2010.

[2] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (abc) algorithm," *Applied soft computing*, vol. 8, no. 1, pp. 687–697, 2008.

[3] I. J. Fister, D. Fister, and X.-S. Yang, "A hybrid bat algorithm," *Electrotechnical review*, vol. 80, no. 1-2, pp. 1–7, 2013.

[4] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[5] I. Fister, I. J. Fister, X.-S. Yang, and J. Brest, "A comprehensive review of firefly algorithms," *Swarm and Evolutionary Computation*, 2013.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948, IEEE, 1995.

[7] I. J. Fister, X.-S. Yang, I. Fister, J. Brest, and D. Fister, "A brief review of nature-inspired algorithms for optimization," *Electrotechnical review*, vol. 80, no. 3, 2013.

[8] O. Holland and C. Melhuish, "Stigmergy, self-organization, and sorting in collective robotics," *Artificial Life*, vol. 5, no. 2, pp. 173–202, 1999.

[9] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Berlin: Springer-Verlag, 2003.

[10] I. Rechenberg, *Evolutionstrategie: Optimirung Technisher Systeme nach Prinzipen des Biologischen Evolution*. Stuttgard: Fromman-Hozlboog Verlag, 1973.

[11] I. Fister, M. Mernik, and B. Filipič, "Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm," *Computer Optimization and Application*, vol. 54, pp. 741–770, Apr. 2013.

[12] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pp. 65–74, 2010.

[13] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver Press, 2011.

[14] X.-S. Yang and X. He, "Bat algorithm: literature review and applications," *International Journal of Bio-Inspired Computation*, vol. 5, no. 3, pp. 141–149, 2013.

[15] G. Wang and L. Guo, "A novel hybrid bat algorithm with harmony search for global numerical optimization," *Journal of Applied Mathematics*, vol. 2013, Article ID 696491, 2013.

[16] Z. W. Geem, J. H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[17] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution a practical approach to global optimization*. Springer-Verlag, 2005.

[18] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009.

[19] I. Fister, "A comprehensive review of bat algorithms and their hybridization," *Master Thesis, University of Maribor, Slovenia*, 2013.

[20] S. Rao, *Engineering optimization : theory and practice*. New York, NY, USA: John Wiley & Sons, Inc., 2009.

[21] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[22] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 6, pp. 646–657, 2006.

[23] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[24] X.-S. Yang, "Appendix a: Test problems in optimization," in *Engineering Optimization* (X.-S. Yang, ed.), pp. 261–266, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010.

[25] X.-S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, pp. 78–84, 2010.

[26] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, pp. 675–701, December 1937.

[27] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, pp. 86–92, March 1940.

[28] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, December 2006.