Performance of a Modified Cuckoo Search Algorithm for Unconstrained Optimization Problems

Milan TUBA
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
tuba@ieee.org

Milos SUBOTIC
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
milos.subotic@gmail.com

Nadezda STANAREVIC
Faculty of Mathematics
University of Belgrade
Studentski trg 16
SERBIA
srna@stanarevic.com

Abstract: - Cuckoo search (CS) algorithm is one of the latest additions to the group of nature inspired optimization heuristics. It has been introduced by Young and Deb in 2009 and was proven to be a promising tool for solving hard optimization problems. This paper presents a modified cuckoo search algorithm for unconstrained optimization problems. We implemented a modification where the step size is determined from the sorted, rather than only permuted fitness matrix. Our modified algorithm was tested on ten standard benchmark functions. Experimental results show that our modification improves results in most cases, compared to the pure cuckoo search algorithm.

Key-Words: - Cuckoo search, Metaheuristic optimization, Unconstrained optimization, Nature inspired algorithms

1 Introduction

Optimization has always been an active research area since many real-world optimization problems belong to a class of hard problems. In all optimization problems the goal is to find the minimum or maximum of the objective function. Thus, unconstrained optimization problems can be formulated as minimization or maximization of *D*-dimensional function:

$$Min\ (or\ max)\ f(x),\ x=(x_1,x_2,x_3,\ldots x_D) \qquad (1)$$

where *D* is the number of variables to be optimized.

Many population based algorithms were proposed for solving unconstrained optimization problems. Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Bee Algorithms (BA) are among most popular such algorithms. The success or failure of a population based algorithms depends on its ability to establish proper trade-off between exploration and exploitation. A poor balance between exploration and exploitation may result in a stagnation and premature convergence to local optima [1], [2].

GA is one of the most popular evolutionary algorithms (EA) in which a population of individuals evolves (moves through the fitness landscape) according to a set of rules such as selection, crossover and mutation [3], [4], [5], possibly hybridized with local search [6]. In GA exploitation is performed by selection operator where individuals move to the peaks in the fitness landscape. Exploration is achieved by perturbing individuals using crossover and mutation operators. There is also fourth operator, introduced by *John Holland*, called inversion which is nowadays rarely used.

PSO algorithm is another example of population based algorithms [7]. PSO is a stochastic optimization technique which is well adapted to the optimization of nonlinear functions in multi-dimensional space and it has been applied to many real-world problems [8]. A basic variant of the PSO algorithm operates by having a population (swarm) of candidate solutions (particles). Particles are moved within the search space according to a simple equation. The movements of the particles are guided by their own best known position in the search space as well as the entire swarm's best known position.

Ant colony optimization is well established swarm intelligence metaheuristic based on ant colony foraging behavior. It has been used successfully for different problems [9], [10], [11].

Several metaheuristics have been proposed to model the specific intelligent behavior of honey bee swarms [12], [13], [14], [15]. The bee swarm

This research is supported by Ministry of Science, Republic of Serbia, Project No. III-44006

intelligence was used in the development of artificial systems aimed at solving complex problems in traffic and transportation [13]. That algorithm is called bee colony optimization metaheuristic (BCO), which is used for solving deterministic combinatorial problems, as well as combinatorial problems characterized by uncertainty. The artificial bee colony (ABC) algorithm is a relatively new population based meta-heuristic approach based on honey bee swarm [16]. In this algorithm possible solution to the problem is represented by food source. Quality of the solution is indicated by the amount of nectar of a particular food source. Exploitation process is carried by employed and onlooker bees, while exploration is done by scouts. There are a number of improvements to the ABC algorithm [17], [18], [19].

A new metaheuristic search algorithm, called cuckoo search (CS), based on cuckoo bird's behavior has been developed by Yang and Deb [20]. It is a very new population based metaheuristic for global optimization and only few papers have been published about it [21], [22]. The original article does not explain all implementation details, there are some differences between descriptions in the original paper and later book by same authors [23]. We wanted to test our understanding of the original algorithm as well as to try to improve it. In this paper we will introduce our modified version of CS algorithm and validate it against pure version on ten standard unconstrained test functions. For testing purposes, we developed our CS software named *CSapp* on which we implemented both, original and modified CS algorithm in order to make comparison.

The rest of this paper is organized as follows. In Section 2, we give a brief introduction to cuckoo bird's behavior which is necessary in order to understand CS algorithm. However, the target topic of Section 2 is detailed description of CS algorithm in both forms: original and modified. Section 3 reports numerical test functions, experimental settings of the algorithm, and experimental analysis on the proposed approach in comparison with the original CS algorithm. Finally, Section 4 concludes this work.

2 Cuckoo search algorithm

Cuckoo birds attract attention because of their unique aggressive reproduction strategy. Cuckoos engage brood parasitism. It is a type of parasitism in which a bird (brood parasite) lays and abandons its eggs in the nest of another species. There are three basic types of brood parasitism: intraspecific brood parasitism, cooperative breeding, and nest takeover [21]. Some species such as the *Ani* and *Guira* cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs [20]. Some host birds do not behave friendly against intruders and engage in direct conflict with them. In such situation host bird will throw those alien eggs away. In other situations, more friendly hosts will simply abandon its nest and build a new nest elsewhere.

While flying, some animals and insects follow the path of long trajectories with sudden 90^0 turns combined with short, random movements. This random walk is called Lévy flight and it describes foraging patterns in natural systems, such as systems of ants, bees, bumbles, and even zooplanktons. Mathematical formulation of Lévy flight relates to chaos theory and it is widely used in stochastic simulations of random and pseudo-random phenomena in nature. These flights can also be noticed in the movements of turbulent fluids. One example of Lévy flight paths is depicted on Fig. 1.

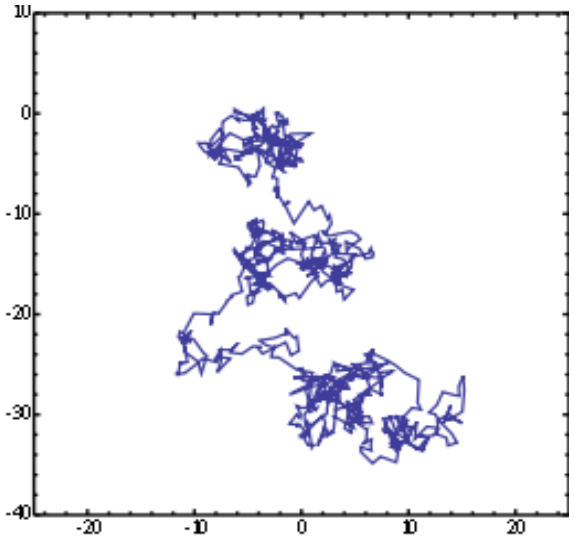

Fig. 1. Possible Lévy flight path

Lévy-style search behavior [24] and random search in general has been applied to optimization and implemented in many search algorithms [25], [26]. One of such algorithms is CS [21]. Preliminary results show its promising capability.

2.1 Description of the original CS algorithm

Cuckoo Search algorithm is population based stochastic global search metaheuristics. It is based on the general random walk system which will be briefly described in this chapter.

In Cuckoo Search algorithm, potential solutions corresponds to Cuckoo eggs. Nature systems are complex and thus, they cannot be modeled by computer algorithms in its basic form. Simplification of natural systems is necessary for successful implementation in computer algorithms.

One approach is to simplify novel Cuckoo Search algorithm through three below presented approximation rules:

- Cuckoos chose random location (nest) for laying their eggs. Artificial cuckoo can lay only one egg at the time.
- Elitist selection process is applied, so only the eggs with highest quality are passed to the next generation
- Host nests number is not adjustable. Host bird discovers cuckoo egg with probability $p_d \in [0,1]$. If cuckoo egg is disclosed by the host, it may be thrown away, or the host may abandon its own nest and commit it to the cuckoo intruder.

To make the things even simpler, the last assumption can be approximated by the fraction of p_d of n nests that are replaced by new nests with new random solutions. Considering maximization problem, the quality (fitness) of a solution can simply be proportional to the value of its objective function. Other forms of fitness can be defined in a similar way the fitness function is defined in genetic algorithms and other evolutionary computation algorithms. A simple representation where one egg in a nest represents a solution and a cuckoo egg represents a new solution is used here. The aim is to use the new and potentially better solutions (cuckoos) to replace worse solutions that are in the nests. It is clear that this algorithm can be extended to the more complicated case where each nest has multiple eggs representing a set of solutions.

A new solution $x^{(t+1)}$ for cuckoo i is generated using a Lévy flight according to the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \wedge Lévy\ (\lambda), \qquad (2)$$

where α (α>0) represents a step scaling size. This parameter should be related to the scales of problem the algorithm is trying to solve. In most cases, α can be set to the value of 1 or some other constant. The product $\wedge$ represents entry-wise multiplications.

Eq. (2) states that described random walk is a *Markov chain,* whose next location depends on two elements: current location (first term in Eq. 2) and transition probability (second term in the same expression).

The random step length is drawn from a Lévy distribution which has an infinite variance with an infinite mean:

$$Lévy \sim u = t^{-\lambda} \qquad (3)$$

where $\lambda \in (0,3]$.

Taking into account basic three rules described above, pseudo code for CS algorithm is:

Start
Objective function $f(x)$, $x= (x_1,x_2...x_u)^T$
Generating initial population of n host nests x_i ($i=1,2,...n$)
While ($t<MaxGenerations$) and (! termin.condit.)
Move a cuckoo randomly via Lévy flights
Evaluate its fitness F_i
Randomly choose nest among n available nests (for example j)
If($F_i > F_j$) Replace j by the new solution;
Fraction p_d of worse nests are abandoned and new nests are being built;
Keep the best solutions or nests with quality solutions;
Rank the solutions and find the current best
End while
Post process and visualize results
End

2.2 Modified cuckoo search algorithm

In the real world, if a cuckoo's egg is very similar to a host's eggs, then this cuckoo's egg is less likely to be discovered, thus the fitness should be related to the difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes.

Both, original, and modified code use random step sizes. Compared to the original code, we use different function set for calculating this step size. In the original code, step size is calculated using following code expression:

*r*nests[permute1[i]][j]- nests[permute2 [i]][j]* (4)

where *r* is random number in [0,1] range, *nests* is matrix which contains candidate solutions along with their variables, *permute1* and *permute2* are different rows permutation functions applied on *nests* matrix.

In order to calculate the step size, instead of Equation (4), we used:

*r*nests [sorted [i]][j] - nests [permute [i]][j]* (5)

The difference is that instead of *permute1*, we used *sorted* function. This function sorts *nests*

matrix by fitness of contained solutions. In this way, higher fitness solutions have slight advantage over solutions with lower fitness. This method keeps the selection pressure (the degree to which highly fit solutions are selected) towards better solutions and algorithm should achieve better results. That does not mean that high fitness solutions will flood population and the algorithm will become stuck in local optimum.

CS algorithm detects best solution X_{best} at the beginning of each iterative step. Also, at this point step scale factor is being calculated using Eq. 6:

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}} \right\}^{1/\beta}, \ \sigma_v = 1 \quad (6)$$

where β denotes Lévy distribution parameter and Γ denotes gamma function.

The evolution of cuckoo i starts with the donor vector v, where $v = x_i^{(t)}$. Step size is being calculated according to Eq. 7.

$$Stepsize^{(t+1)} = 0.01 \frac{u^{(t+1)}}{|v^{(t+1)}|^{1/\beta}} (v - x_{best}) \quad (7)$$

where $u \sim N(0, \sigma_u^2)$ $v \sim N(0, \sigma_v^2)$ are samples from corresponding normal distributions. Sample from Levi distribution is generated by Mantegna's algorithm. For the step size according to Levi distribution we used recommended parameter β=1.5.

Our modified algorithm depends on the best found solution but it is not remembered in any separate memory since the best solution, according to the algorithm, is always retained among current solutions. That means that the process retains Markov property since the next state again depends only on the current state and not the past.

CS algorithm uses some sort of elitism and selection similar to that used in harmony search [27]. However, the randomization is more efficient as the step length is heavy-tailed, and any large step is possible. Also, the number of tuning parameters in CS is less than in GA and PSO, and thus CS can be much easier adapted to a wider class of optimization problems.

3 Experiments

In this section, we show experimental results which validated our modified CS algorithm. As mentioned above, we developed CS software (CSapp), and all tests were run in our testing environment. We developed our software in JAVA programming language. We used the latest JDK (Java Development Kit) version 7 and *NetBeans* IDE (Integrated Development Environment) version 6.9.1, which keeps us up to date with the newest software development concepts.

For testing purposes, we also implemented original version of CS algorithm. We compared results of modified CS algorithm with the original one. Comparison of results is shown in the tables within this section.

3.1 Benchmarks

To test the performance of a modified CS, ten well known benchmark functions are used here for comparison, both in terms of optimum solution after a predefined number of iterations and the robustness of the algorithm i.e. mean value and variance and worst solution for a number of runs. These benchmarks are widely used in evaluating performance of population based methods.

Some of the benchmark functions are unimodal, while others are multimodal. A function is called unimodal if it has only one optimum position. The multimodal functions have two or more local optima.

In order to show how our algorithm performs, we used the following set of functions:

- Ackley
- DixonAndPrice
- Griewank
- Penalized
- Rastrigin
- Schwefel
- Sphere
- Step
- Perm
- Zakharov

Ackley function is a continuous, multimodal function obtained by modulating an exponential function with a cosine wave of moderate amplitude. Originally, it was formulated by Ackley only for the two – dimensional case. It is presented here in a generalized, scalable version. Its topology is characterized by an almost flat (due to the dominating exponential) outer region and a central hole or peak where the modulations by the cosine wave become more and more influential. Formulation:

$$f(x) = -20\exp\left(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) -$$
$$-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi Xi)\right) + 20 + e$$

The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$.

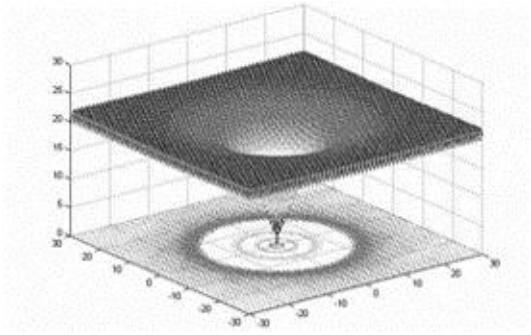

Figure 1: Surface plot for Ackley function

DixonAndPrice is our second test function. Formulation:

$f(x) = (x_1 - 1)^2 + \sum_{I=2}^{n} i(2x_i^2 - x_i - 1)^2$

Global minimum is $f5(x) = 0$.

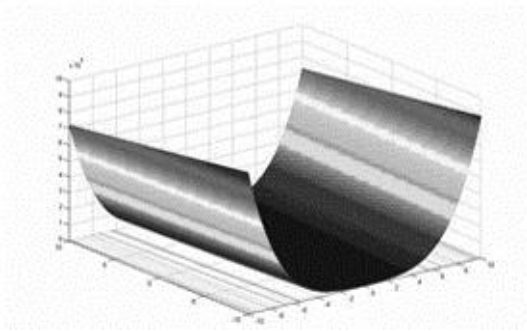

Figure 2: Surface plot for Dixon and Price function

Griewank is third test function. Formulation:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$$

X is in the interval of [-600, 600]. The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (100, 100, \ldots, 100)$. Since the number of local optima increases with the dimensionality, this function is strongly multimodal. The multimodality disappears for sufficiently high dimensionalities (n > 30) and the problem seems unimodal.

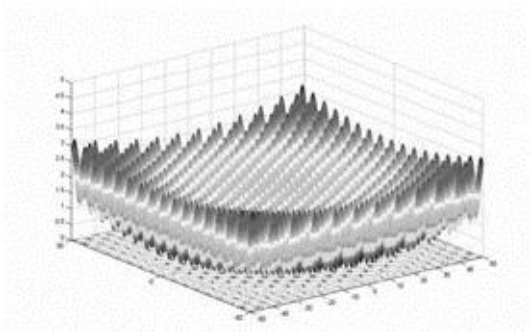

Figure 3: Surface plot for Griewank function

Penalized function is difficult for optimization because of its combinations of different periods of the sine function. Formulation:

$$f(x) = \frac{x}{n}\left\{\begin{matrix} 10\sin^2(\pi y_i) + (y_n - 1)^2 + \\ + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] \end{matrix}\right\} + \\ + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, x_i > a, \\ 0, -a \le x_i \le a \\ k(-x_i - a)^m, x_i < -a \end{cases}$$

The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n\text{-}1, x_n) = (-1, -1, \ldots, -1, -5)$. X is in the interval of [-32, 32].

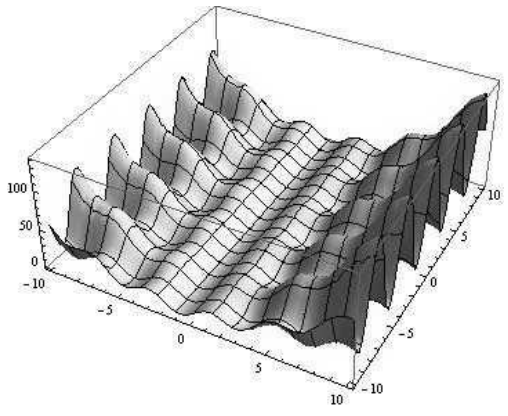

Figure 4: Surface plot for penalized function

Rastrigin function is based on *Sphere* function with the addition of cosine modulation to produce many local minima. Formulation:

$$f(x) = 10n + \sum_{i=1}^{n}(X_i^2 - 10\cos(2\pi x_i))$$

The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$. X is in the interval of [-5.12, 5.12].

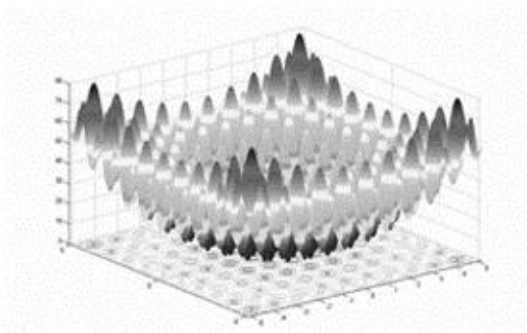

Figure 5: Surface plot for Rastrigin function

Schwefel function as our sixth benchmark. Formulation:

$$f(x) = \sum_{i=1}^{n} -X_i \sin(\sqrt{|X_i|})$$

This function has a value - 418.9828 and at its global minimum (420.9867, 420.9867,…, 420.9867). Schwefel's function is deceptive in that the global minimum is geometrically distant, over the variable space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. Test area is usually restricted to hypercube $-500 \leq xi \leq 500,\ i = 1, \ldots, n$.

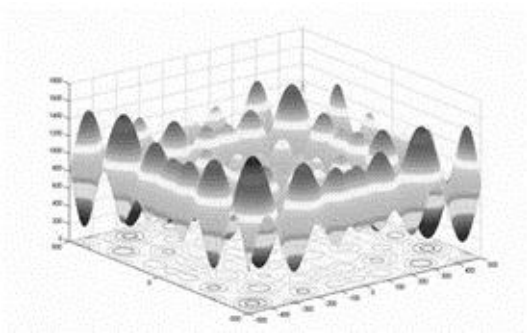

Figure 6: Surface plot for Schwefel function

Sphere function that is continuous, convex and unimodal. Formulation:

$$f(x) = \sum_{i=1}^{n} X_i^2$$

Global minimum value for this function is 0 and optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, .. , 0)$. x is in the interval of [-100, 100].

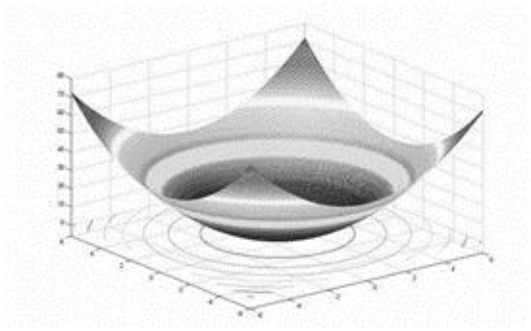

Figure 7: Surface plot for Sphere function

Step is our eighth benchmark function. Formulation:

$$f(x) = \sum_{i=1}^{n}(|xi + 0.5|)$$

This function represents the problem of flat surfaces. It is very hard for algorithms without variable step sizes to conquer flat surfaces problems because there is no information about which direction can provide optimal solution.

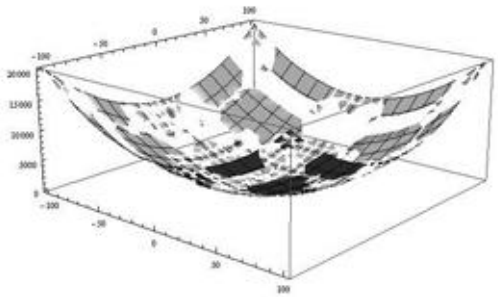

Figure 8: Surface plot for Step function

Ninth function used for tests is ***Perm*** function. Formulation:

$$f(x) = \sum_{k=1}^{n}\left[\sum_{i=1}^{n}(i^k + 0.5)((x_i/i)^k - 1)\right]^2$$

The global minimum of Perm function is 0and corresponding optimal solution is $x_{opt} = (x_1, x_2, \ldots x_n) = (1, 2, \ldots, n)$.

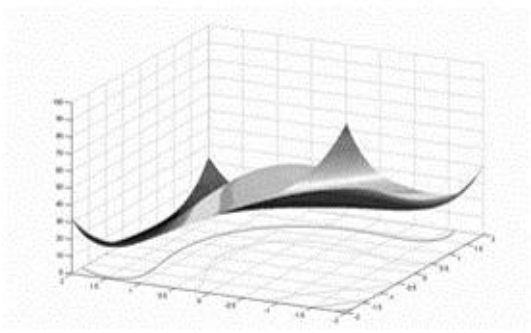

Figure 9: Surface plot for Perm function

Our last test function is ***Zakharov*** function. Formulation:

$$f(x)=\sum_{i=1}^{n}x_i^2+\left(\sum_{i=1}^{n}0.5ix_i\right)^2+\left(i=1\sum_{i=1}^{n}0.5ix_i\right)^4$$

Zakharov function has a global minimum of 0 for $x_{opt} = (x_1, x_2, \ldots x_n) = (0, 0, \ldots, 0)$.

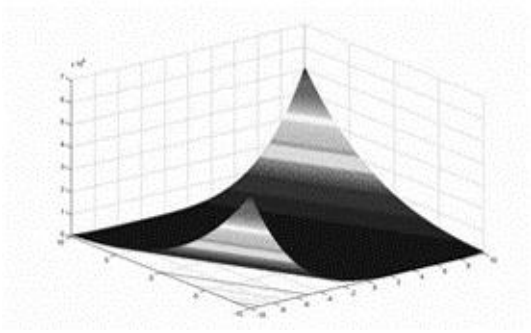

Figure 10: Surface plot for Zakharov function

Summary of above mentioned benchmarks is given in Table 1. Variable range and formulation is shown for the each test function. Optimum for all functions is at *(0, …, 0)*.

3.2 Experimental results and algorithm's settings

We tried to vary the number of host nests (population size *n*) and the probability of discovery p_d. We have used different settings for *n* (5, 10, 15, 20, 50, 100, 150, 250, 500) and for p_d. (0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.4, 0.5) . From test phase simulations, we found that $n = 25$ and $p_d = 0.25$ are sufficient for most optimization problems. This means that the fine parameters tuning are not needed for any given problem. Therefore, we used fixed $n = 25$ and $p_d = 0.25$ for all given problems.

Function	Range	Formulation
Ackley	$[-32,32]^n$	$-20exp\left(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n}X_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi Xi)\right)+20+e$
DixAndPr	$[-10,10]^n$	$(x_1-1)^2+\sum_{I=2}^{n}i(2x_i^2-x_i-1)^2$
Griewank	$[-600,600]^n$	$\sum_{i=1}^{n}\frac{X_i^2}{4000}-\prod_{i=1}^{n}cos(x_i/\sqrt{i})+1$
Penalized	$[-50,50]^n$	$f(x)=\frac{x}{n}\left\{10\sin^2(\pi y_i)+(y_n-1)^2+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]\right\}+\sum_{i=1}^{n}u(x_i,10,100,4)$ $y_i=1+\frac{1}{4}(x_i+1)$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, x_i>a,\\0, -a\le x_i\le a\\k(-x_i-a)^m, x_i<-a\end{cases}$
Rastrigin	$[-5.12,5.12]^n$	$10n+\sum_{i=1}^{n}(X_i^2-10\cos(2\pi X_i))$
Schwefel	$[-500,500]^n$	$\sum_{i=1}^{n}-X_i\sin(\sqrt{\lvert X_i\rvert})$
Sphere	$[-100,100]^n$	$\sum_{i=1}^{n}X_i^2$
Step	$[-100,100]^n$	$\sum_{i=1}^{n}(\lvert xi+0.5\rvert)$
Perm	$[-100,100]^n$	$f(x)=\sum_{k=1}^{n}\left[\sum_{i=1}^{n}(i^k+0.5)((x_i/i)^k-1)\right]^2$
Zakharov	$[-100,100]^n$	$f(x)=\sum_{i=1}^{n}x_i^2+\left(\sum_{i=1}^{n}0.5ix_i\right)^2+\left(i=1\sum_{i=1}^{n}0.5ix_i\right)^4$

Table 1: Benchmark functions summary

We tested each benchmark function six times with 5, 10, 50, 100, 500 and 1000 variables. Thus, we conducted totally 48 tests (8 functions * 6 variable numbers). For every test, we carried on 30 runs with 500 cycles per each run. We printed out best, worst and mean results as well the standard deviation within the set of 30 runs. Parameter settings are in Table 2.

Parameter	Value
Runtime	30
Max Cycle	500
n	25
D	5/10/50/100/500/1000
p_d	0.25

Table 2: Parameter settings for benchmark functions.

Tests were done on Intel Core2Duo T8300 mobile processor with 4GB of RAM on Windows 7 x64 Ultimate Operating System and NetBeans 6.9.1 IDE (Integrated Development Environment).

Experimental results with 5, 10, 50, 100, 500 and 1000 variables are shown in Tables 3, 4, 5, 6, 7 and 8 respectively. All tables have two columns with results in order to make side by side comparison. In the first column, we show results of original, and in the second, results of modified algorithm. We showed values for best, mean, worst and standard deviation.

Function		Original	Modified
Ackley	Best	1.17E-12	3.05E-13
	Mean	8.52E-11	0.01E-11
	Worst	1.24E-09	9.52E-10
	SD	2.23E-10	1.12E-10
DixonAndPrice	Best	3.40E-1	2.32E-2
	Mean	0.057	0.008
	Worst	0.123	0.035
	SD	0.097	0.012
Griewank	Best	1.32E-28	5.66E-29
	Mean	6.72E-26	9.05E-27
	Worst	1.65E-23	5.97E-24
	SD	3.26E-24	9.31E-25
Penalized	Best	1.29E-6	5.49E-7
	Mean	2.04E-5	1.12E-5
	Worst	9.77E-5	5.69E-5
	SD	2.17E-5	0.11E-5
Rastrigin	Best	3.33E-22	1.03E-23
	Mean	1.77E-18	3.52E-19
	Worst	5.32E-16	8.71E-17
	SD	9.56E-16	5.52E-18
Schwefel	Best	0.391	0.352
	Mean	134.333	125.886
	Worst	152.219	138.137
	SD	68.546	56.002
Sphere	Best	5.25E-26	4.63E-28
	Mean	2.50E-22	8.45E-24
	Worst	3.75E-21	7.34E-22
	SD	7.88E-22	4.23E-23
Step	Best	3.87E-6	1.23E-6
	Mean	3.57E-5	1.44E-5
	Worst	4.56E-4	1.002E-4
	SD	4.60E-5	1.05E-5
Perm	Best	1.34E-3	6.20E-4
	Mean	2.89E-3	1.46E-3
	Worst	3.01E-3	1.76E-3
	SD	4.34E-4	7.73E-4
Zakharov	Best	1.02E-1	3.78E-2
	Mean	1.37E-1	4.21E-2
	Worst	1.63E-1	5.98E-2
	SD	1.32E-1	1.58E-2

Table 3: Experimental results for 5 variables

Reported numbers are objective function values, solution vectors are not of particular interest since there are no constraints in these benchmark functions.

As we can see from the Table 3, modified CS algorithm has outperformed the original algorithm in all eight benchmark functions in tests with 5 variables. Best values for almost all test functions are better by factor of 10^{-1}. Difference in mean values is less pronounced. Results for *Sphere* benchmark showed slightly higher difference, where modified CS outscored the original by factor of 10^{-2}.

Function		Original	Modified
Ackley	Best	1.65E-7	3.56E-8
	Mean	1.10E-6	9.13E-7
	Worst	3.85E-6	0.09E-6
	SD	9.30E-7	0.05E-8
DixonAndPrice	Best	0.593	0.455
	Mean	0.664	0.602
	Worst	0.703	0.679
	SD	0.013	0.004
Griewank	Best	9.18E-18	5.67E-19
	Mean	5.03E-15	7.56E-16
	Worst	4.82E-14	9.75E-15
	SD	9.89E-15	8.92E-16
Penalized	Best	3.20E-4	8.45E-4
	Mean	2.19E-4	1.22E-3
	Worst	1.01E-3	4.32E-2
	SD	9.93E-3	1.02E-2
Rastrigin	Best	1.77E-15	5.03E-17
	Mean	4.72E-09	8.51E-11
	Worst	4.54E-08	8.13E-9
	SD	1.16E-08	5.81E-10
Schwefel	Best	661.309	628.205
	Mean	883.518	862.334
	Worst	903.102	891.367
	SD	124.294	102.004
Sphere	Best	4.29E-15	1.09E-16
	Mean	6.04E-13	2.32E-14
	Worst	5.12E-12	4.56E-13
	SD	9.66E-13	4.29E-14
Step	Best	8.12E-5	1.05E-5
	Mean	3.33E-4	1.23E-4
	Worst	2.52E-3	0.05E-3
	SD	7.66E-4	2.52E-4
Perm	Best	5.03E-2	1.06E-2
	Mean	7.12E-2	6.37E-2
	Worst	8.64E-2	8.02E-2
	SD	5.20E-3	4.60E-3
Zakharov	Best	4.77E-1	3.34E-1
	Mean	5.41E-1	4.26E-1
	Worst	6.56E-1	5.34E-1
	SD	1.87E-1	1.78E-1

Table 4: Experimental results for 10 variables

Both algorithms achieved downward results in *Schwefel* function tests, where best results for both algorithms reached are far away from *Schwefel`s* real optimum (0.391 (original) compared to 0.352 (modified)). This can be explained with deceptive nature of *Schwefel* function whose global minimum is geometrically distant over the variables space from the next best local minima. Deceptiveness protracts optimization and algorithms converge in the wrong direction (see Section 3.1).

Function		Original	Modified
Ackley	Best	3.91E-5	0.72E-5
	Mean	5.39E-4	3.21E-4
	Worst	1.25E-3	0.05E-3
	SD	4.19E-4	2.09E-4
DixonAndPrice	Best	0.667	0.625
	Mean	0.674	0.661
	Worst	0.692	0.685
	SD	0.022	0.015
Griewank	Best	1.58E-9	9.92E-10
	Mean	1.60E-8	0.41E-8
	Worst	1.10E-7	9.92E-8
	SD	2.50E-8	1.02E-8
Penalized	Best	0.115	0.193
	Mean	0.206	0.288
	Worst	0.295	0.322
	SD	0.039	0.076
Rastrigin	Best	8.53E-8	2.39E-14
	Mean	8.01E-6	2.39E-10
	Worst	5.48E-5	8.13E-8
	SD	1.23E-5	0.03E-7
Schwefel	Best	9781.549	9653.209
	Mean	10813.984	10012.562
	Worst	10905.389	10652.001
	SD	361.787	307.905
Sphere	Best	2.36E-8	3.02E-10
	Mean	4.64E-6	9.45E-8
	Worst	4.28E-5	3.89E-6
	SD	8.43E-6	9.49E-8
Step	Best	3.392	3.013
	Mean	4.141	3.994
	Worst	4.308	4.215
	SD	0.532	0.486
Perm	Best	3.38E-1	2.24E-1
	Mean	4.16E-1	3.74E-1
	Worst	5.45E-1	4.98E-1
	SD	1.41E-1	1.28E-1
Zakharov	Best	5.441	3.767
	Mean	7.372	6.662
	Worst	8.231	7.567
	SD	2.817	1.705

Table 5: Experimental results for 50 variables

If we compare test results with 5 and 10 variables (see Tables 3 and 4), we can observe that the performance is much worse in tests with 10 variables in the original, as well in the modified CS algorithm.

It is also interesting to notice that the original CS algorithm outperformed modified CS in *Penalized* test. It was not the case in 5 variables test. For other benchmarks, modified CS still performs better than the original.

Now, if we contrast results for 10 variables with 50 variables (see Tables 4 and 5), only minor performance impact can be noticed in higher number of variables test. Exceptions are S*phere, Rastrigin* and *Griewank* functions. For example, best value for *Sphere* of modified algorithm is $1.09*10^{-16}$ in 10 parameter test, while the same value for 50 parameter test is $3.02*10^{-10}$. Performance difference of 10^6 in favor of 10 parameter test is significant.

In *Rastrigin* test, difference in best value is also significant (10^{-3}), while mean value comparison shows just slightly performance variation (10^{-1}).

The greatest dissimilarity shows *Griewank* test. If we compare modified algorithms` results, we can notice that in 10 parameter tests, performance for best, mean and worst values are better by approximately the factor of 10^8 than results in 50 variables test.

Overall, in 50 parameter test, modified CS shows better performance than the original for all benchmarks except for *Penalized* functions in all - best, worst, mean and standard deviation indicators. This noteworthy superiority of modified CS algorithm cannot be neglected.

Experimental results in 100 parameter test are shown in Table 6, which is listed below. As we can see from Table 6, in 100 variables test, performance is further downgraded. *Schwefel* function shows results which are higher than 24 thousand. *Penalized* function values are slightly above 0, but still better in the original than in the modified CS algorithm. Only, *Ackley*, *Griewank*, *Sphere* and *Rastrigin* benchmarks achieved results which are close to 0. *Step* function best value is for example far from optimum, in both, original and modified algorithm (12.905 and 12.302, respectively).

Like in previous test, modified algorithm shows better performance than the original. The only exception is *Penalized* function.

In 500 variables test, Table 7, best values for *Griewank*, *Rastrigin* and *Sphere* functions only reach results close to 0. Also, modified CS obtained best *Ackley`s* result which is near to 0 ($6.78*10^{-1}$).

Function		Original	Modified
Ackley	Best	1.93E-4	0.31E-4
	Mean	0.002	0.002
	Worst	0.003	0.003
	SD	0.002	0.002
DixonAndPrice	Best	0.668	0.598
	Mean	0.698	0.676
	Worst	0.711	0.695
	SD	0.057	0.045
Griewank	Best	3.62E-9	0.61E-9
	Mean	2.81E-8	1.05E-8
	Worst	3.30E-6	1.15E-6
	SD	3.11E-7	1.85E-7
Penalized	Best	0.402	0.502
	Mean	0.474	0.594
	Worst	0.587	0.659
	SD	0.047	0.092
Rastrigin	Best	4.45E-6	1.91E-11
	Mean	1.31E-4	0.06E-8
	Worst	7.00E-4	9.56E-5
	SD	1.64E-4	1.12E-5
Schwefel	Best	24983.536	24339.456
	Mean	26902.073	26233.125
	Worst	27213.152	26886.309
	SD	664.865	605.009
Sphere	Best	3.12E-6	9.83E-8
	Mean	4.62E-5	7.24E-7
	Worst	1.85E-4	6.72E-5
	SD	5.28E-5	6.09E-6
Step	Best	12.905	12.302
	Mean	15.647	15.133
	Worst	18.203	17.898
	SD	0.847	0.701
Perm	Best	7.543	3.574
	Mean	16.964	8.614
	Worst	19.874	11.724
	SD	6.449	4.270
Zakharov	Best	13.895	11.287
	Mean	87.156	67.643
	Worst	102.647	89.083
	SD	81.887	77.021

Table 6: Experimental results for 100 variables

If we observe mean values, it can be noticed that, within all tests conducted on original CS, only for *Griewank's* benchmark mean value is satisfying. On the other side, in modified CS, means for *Rastrigin* and *Sphere* functions are also satisfying. In those cases, modified CS substantially outscores the original. In this case, the difference between the original and modified CS algorithm can be well noticed.

Sphere`s* function best result obtained by modified algorithm is $8.13*10^{-6}$ and such, it is close to real optimum. *Sphere`s mean results, generated by modified CS algorithm, show great performance ($3.45*10^{-2}$). On the other side, mean value for the same function produced with original algorithm is slightly above 0 (0.004).

Function		Original	Modified
Ackley	Best	0.002	6.78E-1
	Mean	0.449	0.222
	Worst	5.419	3.905
	SD	1.257	0.872
DixonAndPrice	Best	0.908	0.832
	Mean	1.049	0.967
	Worst	1.236	1.164
	SD	0.068	0.031
Griewank	Best	3.68E-6	0.15E-6
	Mean	1.07E-4	9.08E-5
	Worst	0.001	9.92E-1
	SD	2.63E-4	8.51E-5
Penalized	Best	0.955	1.019
	Mean	1.012	1.088
	Worst	1.049	1.115
	SD	0.023	0.083
Rastrigin	Best	4.74E-4	5.25E-6
	Mean	0.010	7.15E-2
	Worst	0.085	4.25E-1
	SD	0.017	6.56E-2
Schwefel	Best	171341.672	165236.113
	Mean	174476.232	168199.252
	Worst	176853.623	171562.306
	SD	1320.462	1250.941
Sphere	Best	5.91E-4	8.13E-6
	Mean	0.004	3.45E-2
	Worst	0.014	7.59E-1
	SD	0.004	3.05E-1
Step	Best	112.947	110.135
	Mean	116.062	113.892
	Worst	118.005	115.991
	SD	1.040	0.952
Perm	Best	89.364	78.989
	Mean	123.876	112.958
	Worst	154.747	139.003
	SD	56.909	49.637
Zakharov	Best	768.839	809.876
	Mean	890.983	872.929
	Worst	1022.987	987.928
	SD	125.998	112.987

Table 7: Experimental results for 500 variables

Empirical results for 1000 tests are shown in Table 8.

From Table 8, we can see that original CS algorithm in 1000 variables test obtained satisfying results only in *Griewank* benchmark. For all other test functions, results are less or more greater than 0, which is optimum for all benchmarks.

Modified CS showed different results. Best values for *Ackley*, *Griewank*, *Rastrigin* and *Sphere* for modified CS tests are almost optimal.

Function		Original	Modified
Ackley	Best	0.005	9.99E-1
	Mean	2.426	0.952
	Worst	11.469	3.897
	SD	3.559	1.005
DixonAndPrice	Best	1.022	0.959
	Mean	1.482	1.159
	Worst	3.781	2.013
	SD	0.545	0.302
Griewank	Best	3.82E-5	9.78E-6
	Mean	3.30E-4	0.35E-4
	Worst	0.001	9.98E-1
	SD	3.02E-4	1.05E-4
Penalized	Best	1.069	1.103
	Mean	1.104	1.162
	Worst	1.143	1.198
	SD	0.015	0.095
Rastrigin	Best	0.003	9.03E-1
	Mean	0.032	0.001
	Worst	0.176	0.002
	SD	0.043	0.011
Schwefel	Best	364519.643	352762.115
	Mean	368663.053	357105.879
	Worst	373109.852	361501.081
	SD	1912.757	982.250
Sphere	Best	0.002	7.17E-1
	Mean	0.028	0.001
	Worst	0.112	0.004
	SD	0.027	0.002
Step	Best	238.727	230.651
	Mean	242.256	234.901
	Worst	244.712	236.611
	SD	1.374	1.152
Perm	Best	178.534	156.645
	Mean	203.654	187.536
	Worst	276.982	259.489
	SD	92.332	78.488
Zakharov	Best	1439.165	1423.547
	Mean	1892.524	1765.825
	Worst	2212.165	2182.753
	SD	546.186	543.416

Table 8: Experimental results for 1000 variables

In this, 1000 parameter test, performance residuum between original and modified CS algorithm is the most obvious.

In order to see performance decrement with the rise in number of variables on original and modified CS, we summarized results for *Rastrigin* function. Results for the original CS are presented in Table 9, while the same results are shown for modified CS in Table 10.

	Best	Mean	Worst	SD
D=5	3.33E-22	1.77E-18	5.32E-16	9.56E-16
D=10	1.77E-15	4.72E-09	4.54E-08	1.16E-08
D=50	8.53E-8	8.01E-6	5.48E-5	1.23E-5
D=100	4.45E-6	1.31E-4	7.00E-4	1.64E-4
D=500	4.74E-4	0.010	0.085	0.017
D=1000	0.003	0.032	0.176	0.043

Table 9: *Rastrigin* function results – original CS

Performance difference between original and modified algorithm is less in 5 and 10 parameter tests (compare results in Tables 9 and 10). In these tests, modified CS outscores original by approximately 10^1.

In 50 parameter test, greater result diversity can be noticed. Results difference for best values is 10^6, while means discern by 10^4. Similar situation is in 100 parameter test.

In 500 parameter test, original CS manifests satisfying results for best value only ($4.74*10^{-4}$). Other indicators are above 0. In the same test, modified CS shows good results for all indicators (best, mean, and worst).

	Best	Mean	Worst	SD
D=5	1.03E-23	3.52E-19	8.71E-17	5.52E-18
D=10	5.03E-17	8.51E-11	8.13E-9	5.81E-10
D=50	2.39E-14	2.39E-10	8.13E-8	0.03E-7
D=100	1.91E-11	0.06E-8	9.56E-5	1.12E-5
D=500	5.25E-6	7.15E-2	4.25E-1	6.56E-2
D=1000	9.03E-1	0.001	0.002	0.011

Table 10: *Rastrigin* function results – modified CS

Finally, in 1000 variables test, original CS does not generate satisfying results. Best result achieved is 0.003. At the other side, modified algorithm's best result is $9.03*10^{-1}$. For other indicators, modified CS also shows above 0 performances.

As can be seen from presented tables, for almost all test functions, modified CS has performed slightly better than the original algorithm. The exception is *Penalized* function for which the original outperforms modified in 10, 50, 100, 500 and 1000 parameter tests. Although there is no substantial improvement, presented performance benefit should not be neglected. More about unsuccessful.

Modified algorithm, as well as original, establishes a fine balance of randomization and intensification with small number of control parameters. As for any metaheuristic algorithm, a good balance of intensive local search strategy and an efficient exploration of the whole search space

will usually lead to a more efficient algorithm [21]. On the other hand, there are only two parameters in this algorithm, the population size n, and p_d. Once n is fixed, p_d essentially controls the elitism and the balance of the randomization and local search. Few parameters make an algorithm less complex and thus potentially more generic.

4 Conclusion

In this paper, we presented an improved CS algorithm for unconstrained optimization problems. The capability and the performance of this algorithm was investigated through experiments on well-known test problems. The results obtained by the modified CS algorithm are satisfying.

As can be seen from the comparative analysis between the original and modified CS algorithm for unconstrained optimization problems, new algorithm has performed slightly better in seven of eight benchmark functions. For only one function standard CS algorithm outperformed the modified one. Since cuckoo search is a very new algorithm and there are currently only few papers published about it, the algorithm is still in very early stage of development and initial tuning is necessary before it can be fairly compared to other, more mature algorithms. This paper represents an attempt to stabilize and improve algorithm and in the future in should be compared to other metaheuristics on the same benchmark functions. Future work will also include investigation of the modified CS performance in other benchmark and real-life problems.

References:

[1] Milan Tuba: Swarm Intelligence Algorithms Parameter Tuning, Proceedings of the American Conference on Applied Mathematics (AMERICAN-MATH'12), Cambridge, USA, 2012, pp. 389-394

[2] Milan Tuba: Artificial Bee Colony (ABC) Algorithm Exploitation and Exploration Balance, Proceedings of the 1st International Conference on Computing, Information Systems and Communications (CISCO'12), Singapore, May, 2012, pp.252-257

[3] Mitchell Melanie, *Introduction to Genetic Algorithms*, MIT Press, 1999, p. 158.

[4] Vlahovic, N., Problem solving by AI: Logic programming vs. Genetic algorithms; In: Aurer, B., Kermek, D., editors. Proceedings of the 15th International Conference on Information and Intelligent Systems, September 22-24, 2004, Varazdin, Croatia; Faculty of Organization and Informatics Varazdin, University of Zagreb; Varazdin; 2004. pp. 309-317

[5] Neri F. (2012). "Learning and Predicting Financial Time Series by Combining Evolutionary Computation and Agent Simulation", Transactions on Computational Collective Intelligence, vol. 6, Springer, Heidelberg, vol. 7, pp. 202-221

[6] Moghrabi, "Guided Local Search for Query Reformulation using Weight Propagation", International Journal of Applied Mathematics and Computer Science (AMCS), Vol. 16, No. 4, 537-549, 2006.

[7] Muhammad S. Yousuf, Hussain N. Al-Duwaish, Zakaryia M. Al-Hamouz, *PSO based Single and Two Interconnected Area Predictive Automatic Generation Control*, WSEAS Transactions on System and Control, Vol. 5, Issue 8, 2010, pp. 677-690.

[8] Angel E. Mu˜noz Zavala, Arturo H. Aguirre, Enrique R. Villa Diharce, *Constrained Optimization via Particle Evolutionary Swarm Optimization Algorithm (PESO)*, Proceedings of the 2005 conference on Genetic and evolutionary computation, Article in Press, 2005, doi:10.1145/1068009.1068041, pp. 209-216.

[9] Raka Jovanovic, Milan Tuba: *An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem*, Applied Soft Computing, Vol. 11, Issue 8, Dec. 2011, ISSN 1568-4946, pp. 5360–5366

[10] Raka Jovanovic, Milan Tuba, Dana Simian: *Comparison of Different Topologies for Island-Based Multi-Colony Ant Algorithms for the Minimum Weight Vertex Cover Problem*, WSEAS Transactions on Computers, Issue 1, Volume 9, January 2010, pp. 83-92

[11] Milan Tuba, Raka Jovanovic: *An Analysis of Different Variations of Ant Colony Optimization to the Minimum Weight Vertex Cover Problem*, WSEAS Transactions on Information Science and Applications, Issue 6, Volume 6, June 2009, pp. 936-945

[12] Reza A., Alireza M., Koorush Z., *A novel bee swarm optimization algorithm for numerical function optimization*, Communications in Nonlinear Science and Numerical Simulation, Vol. 15, Issue 10, 2009, pp. 3142-3155.

[13] Teodorovic, D., Dell'Orco M., *Bee colony optimization-a cooperative learning approach to complex transportation problems*,

Proceedings of the 16th Mini - EURO Conference and 10th Meeting of EWGT - Advanced OR and AI Methods in Transportation, 2005, pp. 51-60.

[14] Drias, H., Sadeg, S., Yahi, S, *Cooperative bees swarm for solving the maximum weighted satisfiability problem*, Lecture notes in computer science, Volume 3512, 2005, pp. 318-325.

[15] L. Jiann-Horng, H. Li-Ren, *Chaotic bee swarm optimization algorithm for path planning of mobile robots*, Proceedings of the 10th WSEAS international conference on evolutionary computing, 2009, pp. 84-89.

[16] Behriye Akay, Dervis Karaboga, *A modified Artificial Bee Colony algorithm for real parameter optimization*, Information Sciences, Article in Press, doi:10.1016/j.ins.2010.07.015, 2010.

[17] Nebojsa Bacanin, Milan Tuba: Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators, Studies in Informatics and Control, Vol. 21, Issue 2, June 2012, pp.

[18] Nadezda Stanarevic, Milan Tuba, and Nebojsa Bacanin: Modified artificial bee colony algorithm for constrained problems optimization, International Journal of Mathematical Models and Methods in Applied Sciences, Vol. 5, Issue 3, 2011, pp. 644-651

[19] Ivona Brajevic and Milan Tuba: An upgraded artificial bee colony algorithm (ABC) for constrained optimization problems, Journal of Intelligent Manufacturing, published Online First, Jan 2012, DOI: 10.1007/s10845-011-0621-6

[20] Yang, X. S. and Deb, S., *Cuckoo search via Lévy flights*, in: Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), 2009, pp. 210-214.

[21] Yang, X. S., and Deb, S. *Engineering Optimisation by Cuckoo Search*, Int. J. of Mathematical Modelling and Numerical Optimisation, Vol. 1, No. 4, 2010, pp. 330–343.

[22] Milan Tuba, Milos Subotic, Nadezda Stanarevic: Modified cuckoo search algorithm for unconstrained optimization problems, Proceedings of the European Computing Conference (ECC '11), pp. 263-268, Paris, France, April 2011

[23] Xin-She Yang: Nature-Inspired Metaheuristic Algorithms, Second Edition, Luniver Press, 2011, p. 160

[24] Viswanathan, G. M.; Raposo, E. P.; da Luz, M. G. E.: "Lévy flights and superdiffusion in the context of biological encounters and random searches". *Physics of Life Reviews* **5** (3), 2008, pp. 133–150.

[25] T. Y. Chen, Y. L. Cheng, *Global optimization using hybrid approach*, WSEAS Transactions on Mathematics, Vol. 7, Issue 6, 2008, pp. 254-262.

[26] L. Ozdamar: *A dual sequence simulated annealing algorithm for constrained optimization*, Proceedings of the 10th WSEAS International Conference on applied mathematics, 2006, pp. 557-564.

[27] Geem Z. W., Kim J. H., and Loganathan G. V., A New Heuristic Optimization Algorithm: Harmony Search, Simulation, 2001, vol. 76 No. 2, pp. 60-68