Implementation and performance of an object-oriented software system for cuckoo search algorithm

Nebojsa Bacanin

***Abstract*—**Evolutionary computation (EC) algorithms have been successfully applied to hard optimization problems. In this very active research area one of the newest EC algorithms is a cuckoo search (CS) metaheuristic for unconstrained optimization problems which was developed by Yang and Deb in MATLAB software. This paper presents our software implementation of CS algorithm which we called CSApp. CSApp is an object-oriented system which is fast, robust, scalable and error prone. User friendly graphical user interface (GUI) enables simple adjustment of algorithm's control parameters. The system was successfully tested on standard benchmark functions for unconstrained problems with various number of parameters. CSApp software, as well as experimental results are presented in this paper.

***Keywords*—**Cuckoo search, Metaheuristic optimization, Software system, Nature inspired algorithms.

I. Introduction

MANY practical, real life, problems belong to a class of intractable combinatorial (discrete) or numerical optimization problems because they are often highly nonlinear. Optimization, or mathematical programming, refers to choosing the best element from some set of available alternatives. This means solving problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from an allowed set of values. This formulation, using a scalar, real-valued objective function, is probably the simplest example. The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics.

In order to solve such problems, many methods for continuous optimization and heuristics for discrete problems were developed. Fitness landscape for such problems is multimodal because of its nonlinear nature. Subsequently, local search algorithms such as hill-climbing and its modifications are not suitable, only global algorithms can obtain optimal solutions [1].

Manuscript received December 25, 2011.

The research was supported by the Ministry of Science, Republic of Serbia, Project No. III-44006

N. Bacanin is with the Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: nbacanin@megatrend.edu.rs

Modern metheuristic algorithms (typically high-level strategies which guide an underlying subordinate heuristic to efficiently produce high quality solutions and increase their performance) can be applied to both problem domains [2]. They include population based, iterative based, stochastic, deterministic and other approaches.

Population based algorithms are working with a set of solutions and trying to improve them. By the nature of phenomenon simulated by the algorithm, population based algorithms can be divided into two groups: evolutionary algorithms (EA) and swarm intelligence based algorithms. The most prominent representative of the first group is genetic algorithms (GA). GA is a method for moving a population of candidate solutions through fitness landscape using nature inspired operators: selection, crossover and mutation. But, second group of algorithms is of our particular interest in this paper.

Researchers` attention has been attracted by the collective intelligent behavior of insects or animal groups such as flocks of birds, schools of fish, colonies of ants or bees and groups of other animals/insects. The aggregate behavior of insects or animals is called swarm behavior and the branch of artificial intelligence which deals with the collective behavior of swarms through complex interactions of individuals without centralized supervision component is referred to as swarm intelligence. Swarm intelligence has some advantages such as scalability, fault tolerance, adaptation, speed, modularity, autonomy, and parallelism [3].

Key factors for optimizing capability of swarm intelligence systems are self-organization and division of labor. In such self-organized system, each component (agent) may respond efficiently to local stimuli individually, but they also can act together to accomplish global task via labor division. Entire system is fully adaptive to internal and external changes. Four basic properties on which self-organization rely are: positive feedback, negative feedback, fluctuations and multiple interactions [4]. Positive feedback refers to a situation when and individual recruits other individuals (agents) by some directive. For example, positive feedback is when bees dance in order to lead other bees to a specific food source site. Negative feedback retracts individuals from bad solution for the problem. Fluctuations are random behaviors of individuals in order to explore new states, such as random flights of scouts

in a bee swarm. Multiple interactions are the basis of the tasks to be carried out by certain rules.

A lot of swarm intelligence algorithms have been developed. For example, ant colony optimization (ACO) is a technique that is quite successful in solving many combinatorial optimization problems. The inspiring source of ACO was the foraging behavior of real ants which enables them to find shortest paths between food sources and their nests. While working from their nests to food source, ants deposit a substance called pheromone. Paths that contain more pheromone concentrations are chosen with higher probability by ants than those that contain lower pheromone concentrations. Thus, ants exchange information indirectly by depositing pheromones. This system is called stigmergy, and it is common among many insect societies.

The core philosophy of ACO algorithm involves the movement of an ant colony through different locations which is directed by two local decision policies: pheromone trails and its attractiveness. This algorithm uses two more mechanisms in order to balance exploitation – exploration tradeoff. These are trail evaporation and daemon actions. First mechanism reduces all trail values over time and decreases possibility of getting stuck in local optima. The daemon actions are used to bias a search process from non-local perspective.

Artificial bee colony algorithm (ABC) models intelligent behavior of honey bee swarm. This algorithm produces enviable results in optimization problems. Here, a possible solution to a problem represents a food source (flower). Nectar amount of flower designates the fitness of a solution. There are three types of artificial bees (agents): employed, onlookers and scouts [3]. They all work together in order to gain optimal solution (find appropriate food source). Employed and onlooker bees conduct exploitation process by generating neighborhood solution of a chosen solution. Solution which cannot be improved by employed and onlooker bees within certain number of trials are considered to be exhausted and they are abandoned. Abandoned solutions are replaced with randomly generated solutions. This is exploration process and it is guided by scout bee.

Besides ABC, there are also other algorithms which simulate behavior of bees, such as: bee colony optimization (BCO) and chaotic bee swarm optimization algorithm.

Particle swarm optimization (PSO) algorithm is another example of swarm intelligence algorithms. PSO simulates social behavior of bird flocking or fish schooling. PSO is a stochastic optimization technique which is well adapted to the optimization of nonlinear functions in multidimensional space and it has been applied to several real-world problems. 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 trivial equitation. 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. This process is repeated until the stopping criteria are met or the optimal solution is found.

Improved version of the PSO algorithm is particle swarm inspired evolutionary algorithm (PS-EA) which is a hybrid model of EA and PSO. PS-EA incorporates PSO with heuristics of EA in the population generator and mutation operator while retaining the workings of PSO.

Recently, a novel metaheuristic search algorithm has been developed by Yang and Deb [5]. It is called cuckoo search (CS) algorithm. It has been shown that it is very promising algorithm which could outperform existing algorithms such as PSO.

In this paper, we will present our implementation of CS algorithm. In order to see its robustness and efficiency, we developed object-oriented CS software, named CSApp, for solving combinatorial and numeric optimization problems in JAVA programming language. This software will be in detail presented in this paper as well as testing results on four standard benchmark functions with varying parameters.

II. Cuckoo Behavior

Cuckoos are special because they have many characteristics that differentiate them from other birds. Their major distinguishing feature is aggressive reproduction strategy. 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 [6].

Cuckoos are brood parasites. They use a kind of klepto-parasitism which involves the use and manipulation of host individuals, either of the same (intraspecific brood-parasitism) or different species (interspecific brood-parasitism) to raise the young of the brood parasite. Brood parasitism in which host birds do not behave friendly against intruders, and throw alien eggs away is called nest takeover. On the other hand, less aggressive hosts will simply abandon its nest and build a new nest at other location or will help care for young that are not their own, at the expense of their own reproduction. This type of bird parasitism is known as cooperative breeding.

Some parasite cuckoo species lay eggs that, to the human eye, appear to mimic the appearance of the eggs of their favorite hosts, which hinders discrimination and removal of their eggs by host species. This cuckoos` feature increases their fertility by reducing the probability of their eggs being discovered by the host bird. One example of such behavior is brood-parasitic *Tapera* [6].

In general, cuckoo`s eggs hatch earlier than their host eggs. As soon as the cuckoo chicks have hatched (and before they can even see), they lift any other eggs they find in the nest onto their backs and then throw them overboard. Hatching early means that cuckoo chicks can oust other birds' eggs so that they get all the food their foster parents bring home. Studies also show that a cuckoo chick can also mimic the call of host chicks to gain access to more feeding opportunity [5].

Described cuckoo characteristics, as well behavior model of other animals have widespread use in computational intelligence systems [7].

A. *Lévy flights*

By observing animals foraging behavior, it can be concluded that animals search for food in a random or quasi-random manner. The foraging trajectory of an animal is a random walk because the next step is based on the current location and the probability of moving to the next location.

One type of random walk is Lévy flights in which the step-lengths are distributed according to a heavy-tailed probability distribution. Specifically, the distribution used is a power law of the form $y = x^{-\alpha}$, where $1 < \alpha < 3$, and therefore has an infinite variance. According to conducted studies, foraging behavior of many flying animals and insects show typical characteristics of these flights [8].

Some flies use a series of straight flight paths with sudden 90^0 turn, which leads to a Lévy-flight-style intermittent scale free search pattern [5]. Another example from nature would be when sharks and other sea predators cannot find food for some time. In this situation, they abandon Brownian motion (random motion seen in swirling gas molecules) and their trajectory manifests Lévy-flight – a mix of long and short trajectories, random movements found in turbulent fluids. This data showed that Lévy-flights interspersed with Brownian motion can describe the animals` hunting patterns. Two examples of Lévy-flight path are shown in Fig 1.

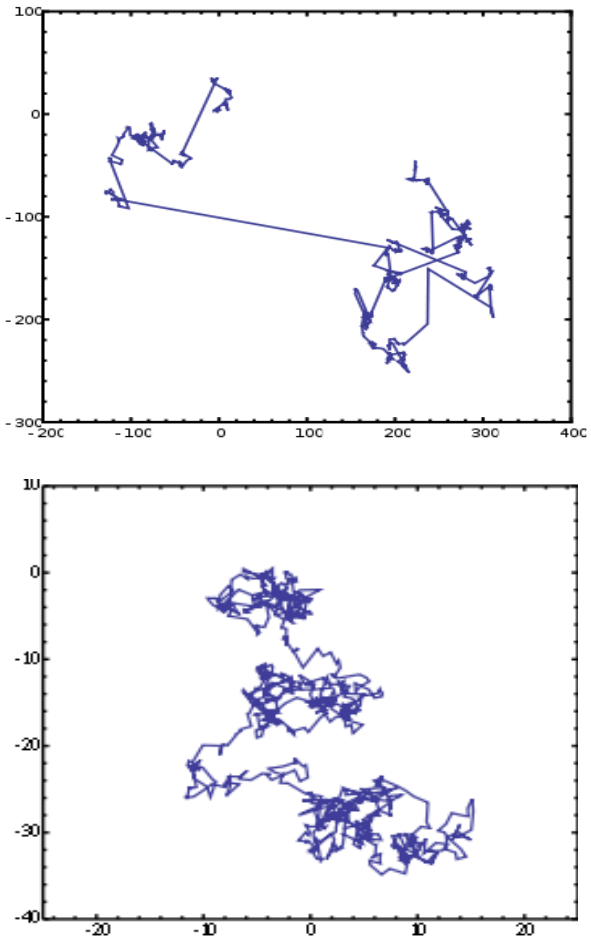

Fig. 1. Lévy flight path example

Besides studies carried on animals, studies on human behavior such as the hunter-gatherer foraging patterns also show the typical feature of Lévy flights. Such behavior has been applied to optimization and optimal search, and preliminary results show its promising capability [5], [6]. Many population based methods use random search similar to Lévy flight [9]

III. Cuckoo search algorithm

In order to simplify the description of novel CS algorithm, three idealized rules can be used [5]:

- Only one egg at a time is laid by cuckoo; Cuckoo dumps its egg in a randomly chosen nest;
- Only the best nests with high quality eggs will be passed into the next generation;
- The number of available host nests is fixed. Egg laid by a cuckoo bird is discovered by the host bird with a probability p_d. In this case, the host bird has two options. It can either throw the egg away, or it may abandon the nest, and build a brand new nest at nearby location.

To make the things even simpler, the last assumption can be approximated by the fraction of p_d of n nests 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 to the fitness function in genetic algorithms and other evolutionary computation algorithms.

A simple representation where one egg in a nest represents a solution and a cuckoo egg represent 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.

When generating new solutions $x^{(t+1)}$ for a cuckoo i, a Lévy flight is performed using the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \wedge L\acute{e}vy\ (\lambda) \qquad (1)$$

where α ($\alpha>0$) represents a step size. This step size should be related to the scales of problem the algorithm is trying to solve. In most cases, α can be set to the value 1. The above expression is in essence stochastic equation for a random walk which is a *Markov chain* whose next location (status) depends on two parameters: current location (first term in Eq. 1) and probability of transition (second term in the same expression). The product $\wedge$ represents entry-wise multiplications. Something similar to entry-wise product is seen in PSO algorithm, but random walk via Lévy flight is much more efficient in exploring the search space as its step length is much longer in the long run [5].

The random step length is drawn from a Lévy distribution

which has an infinite variance with an infinite mean (see equitation 2).

$$Lévy \sim u = t^{-\lambda} \quad (2)$$

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

Here the consecutive jumps (steps) of a cuckoo essentially form a random walk process which obeys a power-law step length distribution with a heavy tail.

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

```
Start
  Objective function f(x), x= (x1,x2...xu)^T
  Generating initial population of n host nests xi
      (i=1,2,...n)
While (t<MaxGenerations) and (! termin.condit.)
  Move a cuckoo randomly via Lévy flights
  Evaluate its fitness Fi
  Randomly choose nest among n available nests
   (for example j)
  If(Fi > Fj) Replace j by the new solution;
  Fraction pd  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
```

At a first glance, it seems that there are some similarities between CS and hill-climbing [10] in respect with some large scale randomization. But, these two algorithms are in essence very different. Firstly, CS is population-based algorithm in a way similar to GA and PSO, but it uses some sort of elitism and/or selection similar to that used in harmony search. Secondly, the randomization is more efficient as the step length is heavy-tailed, and any large step is possible. And finally, the number of tuning parameters is less than in GA and PSO, and thus CS can be much easier adapted to a wider class of optimization problems.

Applications of CS in engineering optimization problems have shown its promising efficiency. For example, for spring design and welded beam design problems, CS achieved better results than other algorithms known in literature. New discrete cuckoo search algorithm has recently been proposed to solve nurse scheduling problem (NSP) [11]. Nurse scheduling concepts are varied due to sophisticated and challenging real world scenarios in nurse management system [11]. Also, an efficient CS based approach has been implemented for data fusion in wireless sensor networks [12]. In this work, Cuckoo Based Particle Approach (CBPA) is used for formulation of optimization network which nodes are deployed randomly and organized as static clusters by CS algorithm.

IV. CSapp software

We have developed our software system for the CS algorithm. It is possible to use existing implementation in *MATLAB 7* [7] which can be found at http://www.mathworks.-com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm, but we chose to develop a new version because we wanted to implement few improvements. Firstly, in order to make algorithm execute faster, we developed our framework. Secondly, our software is object-oriented. With object-oriented concept, software scalability and maintenance is much easier. As an object's interface provides a roadmap for reusing the object in new software, it also provides all the information needed to replace the object without affecting other code. This makes it easy to replace old and aging code with faster algorithms and newer technology. So, if we want to implement new logic for different optimization problems, it will take substantially less time. On the other side, with object-oriented approach, identifying the source of errors becomes easier because objects are self-contained (encapsulation).

We chose to develop CSapp in JAVA programming languages because of its many advantages over C, C++ and other modern programming languages. For example, JAVA does not support features like pointers, multiple inheritance, goto statement and operator overloading, all of which slow application development cycle. Unlike C++, JAVA has garbage collector, so programmers do not need to reallocate the memory or to worry about memory fragmentations. But, the greatest features which motivates us to use JAVA is its portability and JAVA`s APIs (Application Programming Interface). JAVA supports three types of portability: source code portability, CPU architecture portability, and OS/GUI portability. Owing to portability feature, we can use our code on different processors and operating systems with less programming effort.

We used newest JDK (Java Development Kit) version 7 and *NetBeans* IDE (Integrated Development Environment) version 6.9.1, which makes us up to date with the newest software development concepts.

Thus, using previously described environment makes our code more robust, scalable, less error prone and performance is much better.

Screenshot of basic Graphical user interface (GUI) of CSapp while testing Sphere function can be seen in Fig. 2. As we can see form the picture, user can adjust multiple parameters for CS algorithm. Adjustable parameters are:

- *Runtime* defines the number of times to run the algorithm.
- *Max Cycle* defines the number of cycles for improvements. This is a stopping criterion.
- *N* defines the number of nests. Each nest represents one problem solution.
- *D* is the number of parameters of a problem. Functions can be optimized using different set of parameters.
- P_d is discovering probability. This is the probability that a host bird can find an egg laid by cuckoo bird.

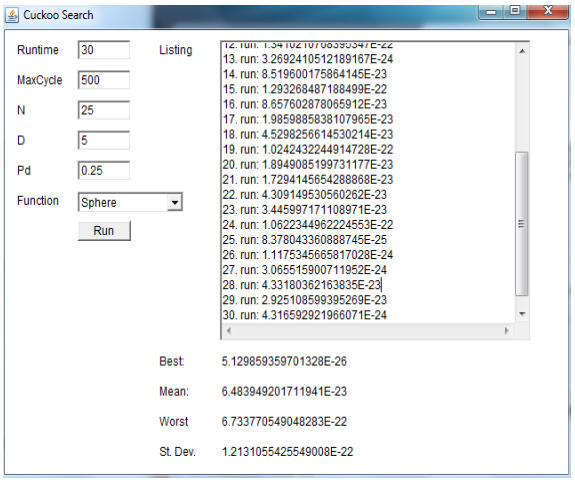

Fig. 2: Screenshot of CSapp GUI while testing Sphere function

In order to show how our software performs, we used four standard benchmark functions [11]:

- Griewank
- Sphere
- Rastrigin
- Ackley

Griewank. The global minimum value for this function is *0* and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$. 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. Because of its interesting surface plot, Griewank is depicted in Fig 3. Definition of the function is:

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

Sphere. The function is continuous, convex and unimodal. Global minimum value 0 and optimum solution is $x_{opt} = (x1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$. Definition:

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

Rastrigin. It is based on *Sphere* function with the addition of cosine modulation to produce many local minima. Thus the function is multimodal. The locations of the minima are regularly distributed. The difficult part about finding optimal solutions to this function is that an optimization algorithm can easily be trapped in a local optimum on its way towards the global optimum The global minimum value for this function is *0* and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$. Definition:

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

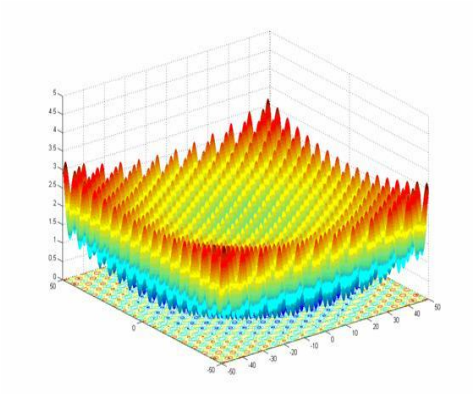

Fig. 3: Surface plot and contour lines of Rastrigin function

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. The global minimum value for this function is *0* and the corresponding global optimum solution is $x_{op}t = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$. Because of its surface plot (Fig. 4), with rise in parameter number, optimization results are getting significantly worse, as will be showed in Section 5. Definition:

$$f(x) = -20\exp(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} X_i^2}) - \exp$$

$$(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi X_i) + 20 + e$$

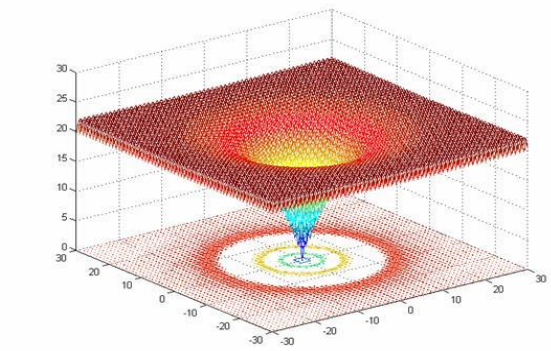

Fig. 4: Surface plot and contour lines of Ackley function

We wanted to see how our software optimizes these four unconstrained benchmark functions. Quick summary of test functions is given in Table 1, while testing results for our algorithm are presented in the following section.

TABLE I
QUICK SUMMARY OF TEST FUNCTIONS

Function	Range	Formulation
Griewank	$[-600,600]^n$	$\sum_{i=1}^{n}\frac{X_i^2}{4000}-\prod_{i=1}^{n}cos(x_i/\sqrt{i})+1$
Sphere	$[-100,100]^n$	$\sum_{i=1}^{n}X_i^2$
Rastrigin	$[-5.12,5.12]^n$	$10n+\sum_{i=1}^{n}(X_i^2-10cos(2\pi X_i))$
Ackley	$[-32,32]^n$	$-20exp\left(-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n}X_i^2}\right)-exp\left(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi Xi)\right)+20+e$

V. TESTS AND RESULTS

CFor all benchmark functions we set the parameters as shown in Table 2. All tests were done on Intel Core2Duo T8300 mobile processor on 2.4 MHZ with 4GB of RAM on Windows 7 x64 Operating System and *NetBeans* 6.9.1 IDE. As mentioned in Section 4, we used newest JDK Version 7 in order to achieve competitive performance.

As can be seen from the Table 2, we tested all functions six times with 5,10, 50, 100, 500 and 1000 parameters respectively. We wanted to see how algorithm performs with extremely low, moderate and high number of parameters. We printed out best, mean and worst results as well the standard deviation for 30 runs with 500 cycles per each run.

TABLE II
PARAMETER VALUES FOR BENCHMARK FUNCTIONS

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

We divided tests into two groups. First group comprises tests with 5,10 and 50 parameters, while second group refers to tests with 100, 500 and 1000 parameters. This division enables easier comparison of obtained results.

In Tables 3, 4 and 5 are shown testing results for *Griewank*, *Sphere*, *Rastrigin* and *Ackley* functions with 5,10 and 50 parameters, respectively. Due to most of the results are small numbers close to zero, we used exponential notation.

TABLE III
RESULTS FOR FUNCTION OPTIMIZATION WITH 5 PARAMETERS

Function		D=5
Griewank	Best	1.32E-28
	Mean	6.72E-26
	Worst	1.65E-23
	Stdev.	3.26E-24
Sphere	Best	5.25E-26
	Mean	2.50E-22
	Worst	3.75E-21
	Stdev.	7.88E-22
Rastrigin	Best	3.33E-22
	Mean	1.77E-18
	Worst	5.32E-16
	Stdev.	9.56E-16
Ackley	Best	1.17E-12
	Mean	8.52E-11
	Worst	1.24E-09
	Stdev.	2.23E-10

As we can see from Table 3, for all test functions CS algorithm gained outstanding results. The best value is acquired for *Griewank* function (10^{-28}), which is very close to the optimum. If we compare test results with 5 and 10 parameters (Table 3 and Table 4), we can see that results with ten parameters for the same number of cycles are somewhat worse but again very close to the optimum (error 10^{-18} compared to 10^{-28}).

TABLE IV
RESULTS FOR FUNCTION OPTIMIZATION WITH 10 PARAMETERS

Function		D=10
Griewank	Best	9.18E-18
	Mean	5.03E-15
	Worst	4.82E-14
	Stdev.	9.89E-15
Sphere	Best	4.29E-15
	Mean	6.04E-13
	Worst	5.12E-12
	Stdev.	9.66E-13
Rastrigin	Best	1.77E-15
	Mean	4.72E-09
	Worst	4.54E-08
	Stdev.	1.16E-08
Ackley	Best	1.65E-07
	Mean	1.10E-06
	Worst	3.85E-06
	Stdev.	9.30E-07

From Table 5, which is listed below, it is interesting to notice that performance penalty when going from 10 to 50 parameters is similar to the one when going from 5 to 10 parameters. Again, all results are very close to optimal value and for some reasonable threshold, for example 10^{-5}, all results are perfect.

TABLE V
RESULTS FOR FUNCTION OPTIMIZATION WITH 50 PARAMETERS

Function		D=50
Griewank	Best	1.58E-9
	Mean	1.60E-8
	Worst	1.10E-7
	Stdev.	2.50E-8
Sphere	Best	2.36E-8
	Mean	4.64E-6
	Worst	4.28E-5
	Stdev.	8.43E-6
Rastrigin	Best	8.53E-8
	Mean	8.01E-6
	Worst	5.48E-5
	Stdev.	1.23E-5
Ackley	Best	3.91E-5
	Mean	5.39E-4
	Worst	1.25E-3
	Stdev.	4.19E-4

In another test set, we used much larger number of parameters. Tests on the same benchmark functions with 100, 500 and 1000 parameters are shown on Tables 6, 7 and 8 respectively.

TABLE VI
RESULTS FOR FUNCTION OPTIMIZATION WITH 100 PARAMETERS

Function		D=100
Griewank	Best	3.62E-9
	Mean	2.81E-8
	Worst	3.30E-6
	Stdev.	3.11E-7
Sphere	Best	3.12E-6
	Mean	4.62E-5
	Worst	1.85E-4
	Stdev.	5.28E-5
Rastrigin	Best	4.45E-6
	Mean	1.31E-4
	Worst	7.00E-4
	Stdev.	1.64E-4
Ackley	Best	1.93E-4
	Mean	0.002
	Worst	0.003
	Stdev.	0.002

If we compare results between last test in first set and first test in second set (with 50 and 100 parameters, Tables 5 and 6), it is interesting to notice that *Griewank's* function best solution in test with 100 parameters is just slightly worse than the best solution in the test with 50 parameters (in both cases, error is 10^{-9}). Despite, doubled numbers of parameters in the second test, results are practically the same. This is also the case in mean results comparison. Only in worst results test, *Griewank* function optimization showed smoothly worse result in 100 parameter test, by factor of 10^{-1}.

For all other test functions, best solutions in 100 parameter test are worse than those in 50 parameter test by approximately the factor of 10^{-2}. Similar situation is in mean and worse results comparison with exception of *Ackley* function, where mean and worst results are penalized by 10^{-4} and 10^{-3} respectively.

As we can see from Table 7, results with 500 parameters are noticeable worse than those with 100 parameters (Table 6). For mean, worst and standard deviation, for all test functions with exception of *Griewank*, exponential inscription of decimal number which represents result is no longer needed. For example, even best result for *Ackley* function is noticeably greater than zero (0.002). For all other functions, best results are still smaller than zero (error 10^{-4} or 10^{-6}).

TABLE VII
RESULTS FOR FUNCTION OPTIMIZATION WITH 500 PARAMETERS

Function		D=500
Griewank	Best	3.68E-6
	Mean	1.07E-4
	Worst	0.001
	Stdev.	2.63E-4
Sphere	Best	5.91E-4
	Mean	0.004
	Worst	0.014
	Stdev.	0.004
Rastrigin	Best	4.74E-4
	Mean	0.010
	Worst	0.085
	Stdev.	0.017
Ackley	Best	0.002
	Mean	0.449
	Worst	5.419
	Stdev.	1.257

Finally, as expected, with 1000 parameters tests (Table 8), satisfying results are obtained only for *Griewank* function whose best is within the value of 10^{-5}. Even for *Griewank*, worst result is noticeable greater than zero (0.001). If we would run tests with more than 1000 parameters, *Griewank* function results would become dissatisfactory. For the remaining three benchmark functions, best, mean and worst values are unsatisfying (in the range of [0.003, 11.469]). *Ackley`s* function worst value is even 11.459, which is far away from real optimum.

According to test results, it can be concluded that *Griewank* function is the easiest for optimization by CSapp, while at the other hand, *Ackley* is the hardest for optimization. Also, for *Ackley* function it is interesting to notice performance penalty with the rise in parameter numbers in all, best, mean, worst and standard deviation results.

TABLE VIII
RESULTS FOR FUNCTION OPTIMIZATION WITH 1000 PARAMETERS

Function		D=1000
Griewank	Best	3.82E-5
	Mean	3.30E-4
	Worst	0.001
	Stdev.	3.02E-4
Sphere	Best	0.002
	Mean	0.028
	Worst	0.112
	Stdev.	0.027
Rastrigin	Best	0.003
	Mean	0.032
	Worst	0.176
	Stdev.	0.043
Ackley	Best	0.005
	Mean	2.426
	Worst	11.469
	Stdev.	3.559

In order to visualize test results changes of best, mean, worst and standard deviation results for *Griewank* and *Ackley* functions, summary of results for both functions are shown in Tables 9 and 10 respectively. From presented tables, it can clearly be seen difference in optimization results between *Griewank* and *Ackley* functions.

TABLE IX
RESULTS SUMMARY FOR GRIEWANK FUNCTION

	Best	Mean	Worst	Stdev
D=5	1.32E-28	6.72E-26	1.65E-23	3.26E-24
D=10	9.18E-18	5.03E-15	4.82E-14	9.89E-15
D=50	1.58E-9	1.60E-8	1.10E-7	2.50E-8
D=100	3.62E-9	2.81E-8	3.30E-6	3.11E-7
D=500	3.68E-6	1.07E-4	0.001	2.63E-4
D=1000	3.82E-5	3.30E-4	0.001	3.02E-4

TABLE X
RESULTS SUMMARY FOR ACKLEY FUNCTION

	Best	Mean	Worst	Stdev
D=5	1.17E-12	8.52E-11	1.24E-09	2.23E-10
D=10	1.65E-07	1.10E-06	3.85E-06	9.30E-07
D=50	3.91E-5	5.39E-4	1.25E-3	4.19E-4
D=100	1.93E-4	0.002	0.003	0.002
D=500	0.002	0.449	5.419	1.257
D=1000	0.005	2.426	11.469	3.559

Sphere and *Rastrigin* functions` optimization showed similar results for all test cases. In some tests *Sphere* is slightly better, while in others *Rastrigin* obtained better results. For easier comparison, collectively results (best, mean, worst and standard deviation values) for *Sphere* and *Rastrigin* functions are put in Tables 11 and 12, respectively.

TABLE XI
RESULTS SUMMARY FOR SPHERE FUNCTION

	Best	Mean	Worst	Stdev
D=5	5.25E-26	2.50E-22	3.75E-21	7.88E-22
D=10	4.29E-15	6.04E-13	5.12E-12	9.66E-13
D=50	2.36E-8	4.64E-6	4.28E-5	8.43E-6
D=100	3.12E-6	4.62E-5	1.85E-4	5.28E-5
D=500	5.91E-4	0.004	0.014	0.004
D=1000	0.002	0.028	0.112	0.027

As we can see from Tables 11 and 12, in almost all testing scenarios (different parameter set), result difference between corresponding best, mean and worst values is less than 10^{-1}. The only exception is test with 5 parameter tests, where Sphere function substantially outperformed Rastrigin (relative 10^{-4}).

TABLE XII
RESULTS SUMMARY FOR RASTRIGIN FUNCTION

	Best	Mean	Worst	Stdev
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

VI. CONCLUSION

We designed, developed and tested a software system in JAVA for unconstrained optimization problems based on Yan and Deb`s CS algorithm. Because of its flexible object-oriented design, our software can be modified to accommodate large number of different optimization problems in both, function optimization and engineering domain. Benchmark tests show that superior results can be generated and system is ready to be applied to new problems and used as a valuable tool for further research.

REFERENCES

[1] Yang, X. S., Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008, pp. 128.
[2] T. Y. Chen, Y. L. Cheng, Global optimization using hybrid approach, WSEAS Transactions on Mathematics, Vol. 7, Issue 6, 2008, pp. 254-262.
[3] 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.
[4] 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.
[5] 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.

[6] Yang, X.S., and Deb, S. Engineering Optimization by Cuckoo Search, Int. J. of Mathematical Modeling and Numerical Optimization, Vol. 1, No. 4, 2010, pp. 330–343.
[7] Tricia Rambharose, Alexander Nikov, Computational intelligence-based personalization of interactive web systems, WSEAS Transactions on Information Science and Applications, Vol. 7, Issue 4, 2010, pp. 484-497.
[8] Pavlyukevich I. J. ,Cooling down Lévy flights, J. of Physics A: Mathematical and Theoretical, Vol. 40, No. 41, 2007, pp. 225-232.
[9] M. Ettaouil, C. Loqman: Constraint satisfaction problems solved by semidefinite relaxations, WSEAS Transactions on Computers, Vol. 7, Issue 9, 2008, pp. 951-961.
[10] Mahmood M. Nesheli, Othman C., Arash Moradkhani R., Optimization of Traffic Signal Coordination System on Congestion: A Case Study, WSEAS Transactions on Advances in Engineering Education, Vol. 6, Issue 7, 2009, pp. 203-212.
[11] Tein L. H. and Ramli R., Recent advancements of nurse scheduling models and a potential path, in: Proc. 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA), 2010, pp. 395-409.
[12] M. Dhivya, Energy Efficient Computation of Data Fusion in Wireless Sensor Networks Using Cuckoo Based Particle Approach (CBPA), Int. J. of Communications, Network and System Sciences, Vol. 4, No. 4, 2011, pp. 249-255.
[13] Dervis Karaboga, Bahriye Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization, Vol. 39, Issue 3, 2007, pp. 459-471.

Nebojsa Bacanin received B.S. and M.S. in economics and computer science in 2006 and 2008 from Megatrend University of Belgrade and also M.S. in computer science in 2008 from University of Belgrade

He is currently a Ph.D. student at Faculty of Mathematics, Computer Science department, University of Belgrade and works as teaching assistant at Faculty of Computer Science, Megatrend University of Belgrade. He is the author or coauthor of five papers. His current research interest includes nature inspired metaheuristics.

Mr. Bacanin participated in WSEAS conferences**.**