An object-oriented software implementation of a novel cuckoo search algorithm

Nebojsa BACANIN
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
nbacanin@megatrend.edu.rs

Abstract: - This paper presents an object-oriented software system that implements a cuckoo search (CS) metaheuristic for unconstrained optimization problems. Yang and Deb developed cuckoo search algorithm in MATLAB and tested it on some standard benchmark functions as well as on some engineering optimization problems where it showed promising results. We developed our algorithm in JAVA programming language which is faster and easier for maintenance since it is object-oriented. The application includes user friendly graphical user interface (GUI) and it was successfully tested on standard benchmark functions for unconstrained problems.

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

1 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].

Modern metaheuristic 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). Second group 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.

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). 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. 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 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 standard benchmark functions.

2 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 kleptoparasitism 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. 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 cuckoo’s 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].

2.1 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° turn, which leads to a Lévy-flight-style intermittent scale free search pattern [5] (Fig. 1).

![Fig. 1. Lévy flight path example](image)

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].

3 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 more simple, 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 $x_i^{(t+1)}$ for a cuckoo $i$, a Lévy flight is performed using the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \cdot \text{Lévy} (\lambda)$$  \hspace{1cm} (1)

where $\alpha$ ($\alpha > 0$) represents a step size. This step size should be related to the scales of problem the algorithm is trying to solve. In most cases, $\alpha$ 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 equation 2).

$$\text{Lévy} \sim u = t^\lambda$$  \hspace{1cm} (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 $f(x) = (x_1, x_2, \ldots, x_d)^T$
- Generating initial population of $n$ host nests $x_i (i=1,2,\ldots,n)$
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

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.

4 CSapp software
We have developed our software system for the CS algorithm. It is possible to use existing implementation in MATLAB 7 (R14) 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. We used newest JDK (Java Development Kit) version 7 and NetBeans IDE (Integrated Development Environment) version 6.9.1.

Thus, using previously described environment makes our code more robust, errorless and performance is much better.

Screenshot of basic Graphical user interface (GUI) of CSapp while testing Sphere function can be seen in Fig. 1. 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. 1: Screenshot of CSapp GUI while testing Sphere function](image_url)
In order to show how our software performs, we used four benchmark functions [11]:

- **Griewank**
- **Sphere**
- **Rastrigin**
- **Ackley**

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

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

**Sphere.** The function is continuous, convex and unimodal. Global minimum value for this function is 0 and optimum solution is \( x_{opt} =(x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0) \). 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. 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:

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

**Ackley** function is a continuous, multimodal function obtained by modulating an exponential function with a cosine wave of moderate amplitude. 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:

\[
f(x)= - 20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e
\]

We wanted to see how our software optimizes these four unconstrained benchmark functions. Testing results for our algorithm are presented in the following section.

### 5 Tests and results

For all benchmark functions we set the parameters as shown in Table 1. Tests were done on Intel Core2Duo T8300 mobile processor with 4GB of RAM on Windows 7 x64 Operating System and NetBeans 6.9.1 IDE.

As can be seen from the Table 1, we tested all four functions three times with 5,10 and 50 parameters respectively. We printed out best, mean and worst results as well the standard deviation for 30 runs with 500 cycles per each run.

### Table 1: Parameter values for benchmark functions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>30</td>
</tr>
<tr>
<td>Max Cycle</td>
<td>500</td>
</tr>
<tr>
<td>N</td>
<td>25</td>
</tr>
<tr>
<td>D</td>
<td>5/10/50</td>
</tr>
<tr>
<td>( P_d )</td>
<td>0.25</td>
</tr>
</tbody>
</table>

In Tables 2, 3 and 4 are shown testing results for **Griewank**, **Sphere**, **Rastrigin** and **Ackley** functions with 5,10 and 50 parameters, respectively.

### Table 2: Results for function optimization with 5 parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>D=5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Griewank</strong></td>
<td>Best: 1.32E-28, Mean: 6.72E-26, Worst: 1.65E-23, Stdev: 3.26E-24</td>
</tr>
<tr>
<td><strong>Sphere</strong></td>
<td>Best: 5.25E-26, Mean: 2.50E-22, Worst: 3.75E-21, Stdev: 7.88E-22</td>
</tr>
<tr>
<td><strong>Rastrigin</strong></td>
<td>Best: 3.33E-22, Mean: 1.77E-18, Worst: 5.32E-16, Stdev: 9.56E-16</td>
</tr>
<tr>
<td><strong>Ackley</strong></td>
<td>Best: 1.17E-12, Mean: 8.52E-11, Worst: 1.24E-09, Stdev: 2.23E-10</td>
</tr>
</tbody>
</table>

### Table 3: Results for function optimization with 10 parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>D=10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Griewank</strong></td>
<td>Best: 9.18E-18, Mean: 5.03E-15, Worst: 4.82E-14, Stdev: 9.89E-15</td>
</tr>
<tr>
<td><strong>Sphere</strong></td>
<td>Best: 4.29E-15, Mean: 6.04E-13, Worst: 5.12E-12, Stdev: 9.66E-13</td>
</tr>
<tr>
<td><strong>Rastrigin</strong></td>
<td>Best: 1.77E-15, Mean: 4.72E-09, Worst: 4.54E-08, Stdev: 1.16E-08</td>
</tr>
<tr>
<td><strong>Ackley</strong></td>
<td>Best: 1.65E-07, Mean: 1.10E-06, Worst: 3.85E-06, Stdev: 9.30E-07</td>
</tr>
</tbody>
</table>

Table 2: Results for function optimization with 5 parameters

Table 3: Results for function optimization with 10 parameters

Table 4: Results for function optimization with 50 parameters
As we can see from Table 2, for all test functions CS algorithm gained outstanding results. If we compare test results with five and ten parameters (Table 2 and Table 3), 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>
<thead>
<tr>
<th>Function</th>
<th>D=50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Griewank</td>
<td>Best: 1.58E-9 Mean: 1.60E-8 Worst: 1.10E-7 Stdev: 2.50E-8</td>
</tr>
<tr>
<td>Sphere</td>
<td>Best: 2.36E-8 Mean: 4.64E-6 Worst: 4.28E-5 Stdev: 8.43E-6</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>Best: 8.53E-8 Mean: 8.01E-6 Worst: 5.48E-5 Stdev: 1.23E-5</td>
</tr>
<tr>
<td>Ackley</td>
<td>Best: 3.91E-5 Mean: 5.39E-4 Worst: 1.25E-3 Stdev: 4.19E-4</td>
</tr>
</tbody>
</table>

Table 4: Results for function optimization with 50 parameters

From Table 4 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.

6 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. 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: