# Parallelized Cuckoo Search Algorithm for Unconstrained Optimization

Milos SUBOTIC [1], Milan TUBA [2], Nebojsa BACANIN [3], Dana SIMIAN [4]
[1,2,3] Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29, N. Belgrade
SERBIA
[4] Department of Computer Science
Lucian Blaga University of Sibiu
5-7 dr. I. Ratiu str., Sibiu
ROMANIA
[1] milos.subotic@gmail.com, [2] tuba@ieee.org, [3] nbacanin@megatrend.edu.rs, [4] dana.simian@ulbsibiu.ro

*Abstract:* - Modifications that introduce parallelization of standard cuckoo search algorithm are proposed in this paper. Basic form of the cuckoo search algorithm has already shown great potential for optimization problems, especially when applied to unconstrained continuous functions. In this paper two aspects of parallelization are proposed. The first one addresses the performance issue, while the second one deals with quality of results. Multicore processors became standard today. When different runs of algorithm execute within different threads, better performance can be reached. Second issue refers to multiple flocks approach that combines search results from two or more flocks. Set of well-known unconstrained continuous benchmark function is used to illustrate testing results of the proposed parallelized cuckoo search algorithm.

*Key-Words:* - Cuckoo search, Parallelization, Unconstrained optimization, Swarm intelligence, Metaheuristics

## 1 Introduction

Optimization has been interesting topic for researches over decades. Mathematical optimization refers to choosing the best element from the collection of available elements. The most straightforward case of optimization is minimization or maximization of real function.

If variables of the function that is to be optimized are considered, optimization can be discrete or continuous. The first is also known as combinatorial optimization because we are looking for an array or set of discrete objects such as integers, permutations or graphs that improve objective function value. Continuous optimization can use real values. Continuous optimization problems are widespread in mathematical modeling of real systems and have been successfully applied in many practical applications. Numerical optimization can be further divided into two groups: unconstrained and constrained. In unconstrained optimization no limitations on the values of the parameters exist. Under such condition, an objective function needs to be minimized or maximized:

$$min\ f(x) \quad \text{or} \quad max\ f(x) \tag{1}$$

where $x \in R^n$ is a real vector with $d \geq 1$ components, and $f: R^n \rightarrow R$ is an objective function.

If there are constraints on parameter values, the optimization problem is referred to as constrained. These constraints make certain values illegal that might otherwise be the global optimum. Constrained problems can be defined as follows [1]:

$$min\ f(x),\ x=(x_1, \ldots, x_n) \in R^n \tag{2}$$

where $x \in F \subseteq S$ is a real vector with $d \geq 1$ components, $S \in R^n$. $R^n$ is defined by lower and upper bounds for variables:

$$lb_i \leq x_i \leq ub_i, \quad 1 \leq i \leq n \tag{3}$$

and the feasible region $F \subseteq S$ is defined by a set of $m$ linear or nonlinear constraints:

$$g_j(x) \leq 0, for\ j = 1, \ldots, q$$
$$h_j(x) = 0, for\ j = q + 1, \ldots, m \tag{4}$$

where $q$ is the number of inequality constraints and $m-q$ is the number of equality constraints.

Many optimization methods have been developed. Among the latest are metaheuristics that are designed to deal with complex optimization problems where other optimization methods have failed to be either effective or efficient [2]. Metaheuristics can be applied to a wide variety of practical problems where they achieve respective results. With the progress of metaheuristics such as tabu search, genetic algorithms and simulated annealing, the primary challenge has become the adaptation of metaheuristics to a specific problem. That adaptation is usually much easier than

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

developing a specialized heuristic for specific problem, which makes metaheuristics appealing choice for implementation in general applications [2]. They can successfully be applied to continuous as well to discrete optimization problems [3].

Population based metaheuristics is a class of algorithms that work with a population of possible solutions trying to improve them in an iterative process. Swarm intelligence is a subclass of population based metaheuristics that is inspired by animal colony systems such as flock of birds, school of fish, colony of ants, bee hive or cuckoos. Swarm intelligence can be defined as artificial intelligence based system which relays on the collective behavior of decentralized, self-organized natural systems [4]. Swarm intelligence system is composed of many homogenous components called artificial agents. Local interaction between agents is based on simple rules. Collectively, agents produce complex interactions and behavior which lead whole system to the desired result. Swarm intelligence is sometimes used in conjunction with other algorithms, for example with genetic algorithms (GA) [5].

One of the pioneers of swarm intelligence algorithms is particle swarm optimization (PSO) proposed by Kennedy and Eberhart [6]. This algorithm models social behavior of fish schooling or bird flocking and can search very large spaces of candidate solutions.

Artificial bee colony (ABC) algorithm is metaheuristics that simulates behavior of honey bee swarm [7]. In this algorithm, three types of bees carry on the search process: employed bees, onlookers and scouts. ABC algorithm, as well as its modifications, has been proven very successful with constrained optimization problems [8], [9].

Cuckoo search (CS) is a novel optimizing algorithm developed by Yang and Deb [10]. It is one of the latest swarm intelligence algorithms and has not been researched enough yet [11]. We investigate in this paper how CS algorithm behaves in multithreaded environment. We present parallelized implementation of CS algorithm and how this multithreaded adaptation of CS metaheuristics show substantial improvement over traditional CS.

This paper consists of four sections. Necessary theoretical discussion about optimization and metaheuristics is in Section 1. Section 2 is the core of our paper, where we show both, traditional CS algorithm and its parallelized implementation. Section 3 contains results of numerical experiments where we tested our algorithm on continuous unconstrained optimization problems. Finally, Section 4 concludes this paper.

# 2 Original and Parallelized CS Algorithm

Cuckoo's behavior in nature was inspiring source for developing CS algorithm. Cuckoos are characterized by aggressive reproduction strategy. Females often use another species nests for laying fertilized eggs. In that way, another species raise cuckoo's brood. This phenomenon is called brood-parasitism and it manifests in different forms. If the host belongs to the same species as intruder, it is the form of intraspecific brood-parasitism. Otherwise, it is kind of interspecific brood-parasitism.

If host bird discovers intruder's eggs, it will either throw them away or abandon its own nest and move away. Some parasite cuckoo species lay eggs that mimic the appearance of eggs of their favored hosts, thus avoiding 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 [12]. Cuckoo's chicks can also mimic the call of host's chick. This enables them to gain more food from the host parent. Timing of laying eggs by cuckoos is also very important factor. Cuckoos hatch their eggs earlier than hosts. Following aggressive approach, they evict host eggs by throwing them out of the nest. This increases the cuckoo chick's share of food provided by its host bird.

In order to construct CS algorithm, understanding of cuckoo's foraging behavior is also necessary. The foraging route of cuckoos is random walk because the next step is based on the current location and the probability of moving to the next location. Actually, cuckoos use special type of random walk - Lévy flights, where step-lengths are distributed according to a heavy-tailed probability distribution [13]. Specifically, the distribution used is a power law of the form $y = x^{-\alpha}$, where $1 < \alpha < 3$, and therefore has an infinite variance. Many flying animals use this type of foraging trajectory.

## 2.1. Original CS Algorithm

For successful implementation of CS algorithm, three idealized rules must be applied [14]:

- 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.

ISBN: 978-1-61804-089-3

- The number of available host nests is fixed. Egg laid by a cuckoo bird is discovered by the host bird with a probability $p_d \in [0,1]$. In this case, the host bird has two options. It can either throw the egg away, or it may abandon the nest.

Last rule can be approximated by the fraction of $p_d$ of $n$ nests that are replaced by new nests with new random solutions. Here, we use simple representation where one egg in a nest models a solution, and a cuckoo egg represents a new solution. Goal is to use the new and potentially better solutions to replace worse solutions (eggs in the nests).

The process of generating new solutions $x^{(t+1)}$ for a cuckoo $i$, is performed with Lévy flight according to the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \wedge Lévy\ (\lambda), \quad (5)$$

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, α can be set to the value of 1. The product ^ represents entry-wise multiplications. Described random walk via Lévy flight used here efficient in exploring the search space as its step length is much longer in the long run.

The random step length is drawn from a Lévy distribution which has an infinite variance with an infinite mean [14]:

$$Lévy \sim u = t^{-\lambda} \quad (6)$$

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

Using presented theoretical background, pseudo code for CS algorithm can be summarized as:

**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. Parallelized CS Algorithm

Ideal processor for programing has one very fast core, but in reality, it is very hard to build such CPU. In real life it is much easier and chipper to produce processor with $n$ cores, each of speed $k$, than one core of speed $n * k$.

In last 5 years, processors with multiple cores have become very popular. They are much cheaper, and have less power consumption. Bio inspired search algorithms ware always suitable for parallelization, but with multicore processors paradigm, implementation of parallelized population based algorithms increased. Many of population based algorithm ware parallelized, such as genetic algorithms (GA) [15], ACO [16], [17], PSO [18] and ABC [19]. Parallelization of these algorithms showed great successes, especially for continuous optimization problems, both constrained and unconstrained.

The aim of parallelization is to cut down execution time of algorithm, to obtain better results, or both. Population based algorithm should be run more than once, because they are not deterministic. Final result is usually the best result from multiple runs. Hence the execution of population based algorithm takes some time. Running those algorithms in parallel can be very productive.

Methods that can speed the execution basically are trying to run evaluation function simultaneously for as many individuals as possible. The idea that comes first is to run calculation of objective function for every individual in population in separate thread. This approach showed some significant shortcomings. Population based algorithms usually have between 20 and 100 individuals in population, and managing such large number of threads usually exceeds CPU costs of serial execution of algorithm. In this paper we propose a different approach. Every run of the algorithm is presented as a separate thread, and threads have no communication between themselves at all. CS algorithm is very suitable for this kind of parallelization. Every iteration of algorithm runs as separate thread with different random seed. After all threads are over, best, mean and worst solution is calculated. The speed increases almost as many times as there are execution cores in system, but if only one iteration of algorithm is executing, or algorithm is running on single core system, there is no speed gain at all.

Other way of parallelization of bio inspired heuristics is to try to run more than one population on same search space and to find better ratio between exploration and exploitation. This is usually done by dividing main population in certain

number of subpopulations. This method is used in our algorithm. CS algorithm can achieve good results even with small number of cuckoos. Therefore, it can be divided in 4 sub-flocks easily. Every sub flock is running standard CS algorithm on the same search space with different random seed. After certain number of generations, the results from all flocks are copied into one array, that array is sorted by the quality of the results, and top quarter of array is copied back to flocks. The flocks then continue to execute standard CS algorithm with these results as input. The sub-flocks continue their search from best results from all sub-flocks as starting point. Our experiments showed that best results are obtained when synchronization is performed after 500 to 1000 cycles. This method prevents trapping into local optimum.

# 3 Experiments

In this section, we show experimental results which validate performance of our parallelized CS approach.

## 3.1 Benchmarks

For testing purposes, we used four standard benchmark functions for unconstrained continuous optimization:

- Ackley
- Sphere
- Rastrigin
- Griewank

*Ackley* function is a continuous, multimodal function obtained by modulating an exponential function with a cosine wave of moderate amplitude. 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 $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$.

*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 $x_{opt} = (x_1, x_2, \ldots, x_n) = (0, 0, .. , 0)$.

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

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

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

*Griewank* is third test function. Definition:

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

The global minimum value for this function is 0 and the corresponding global optimum solution is $x_{opt} = (x_1, x_2, \ldots, x_n) = (100, 100, \ldots, 100)$.

## 3.2 Parameter Settings and Testing Results

All of the parallelization approaches have been implemented using Java programming language. In the Java programming language, concurrent programming is mostly concerned with threads. Threads are sometimes called lightweight processes. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process. For test purposes, we created test application in Java programming language based on Yang's and Deb's software in Matlab. Tests were done on Intel I7-2600k processor with 8GB of RAM on Windows 7 x64 Operating System and NetBeans 7.0.1 IDE. The parameters of algorithm are given in table 1.

| Parameter | Value |
|---|---|
| Number of function evaluation calls | 500,000 |
| Colony size | 40 |
| $p_d$ | 0.25 |
| Number of runs with different seeds | 30 |
| Number of function parameters | 5,10,50,500 |

**Table 1**: Parameter settings

In Table 2, results for speed improvements are illustrated. Comparison is shown against standard version of CS algorithm, and it is given for different number of parameters of objective function. The results show the execution time for 30 runs given in seconds.

There is noticeable decrease in execution times for almost any combination of objective function and number of parameters. Since Intel I7-2600K processor has 4 physical and 4 logical cores, the ideal case would be the one in which parallel execution of algorithm is running 4 to 8 time faster than serial one. On this system in same cases parallel runs are executing more than 6 times faster.

ISBN: 978-1-61804-089-3

When algorithm is used for objective functions with smaller number of parameters or for fairly simple objective functions, like *Sphere* function, more CPU resources is used for creating and synchronizing threads than for serial runs of CS algorithm. This is especially noticeable when Sphere function is used for 5, 10 and 50 parameters. Speed gains are starting to show only when number of parameters is set to 500. Since this method of parallelization has no impact on quality of results, comparison of results are omitted.

| Function | Number of parameters | Serial runs (seconds) | Parallel runs (seconds) | Speed increase |
|---|---|---|---|---|
| Ackley | 5 | 108,1 | **106,5** | **1,013786** |
| | 10 | 159,4 | **97,0** | **1,638939** |
| | 50 | 345,8 | **57,9** | **5,963531** |
| | 500 | 2274,0 | **387,2** | **5,873623** |
| Griewank | 5 | 153,7 | **112,6** | **1,358766** |
| | 10 | 195,1 | **98,8** | **1,974645** |
| | 50 | 603,2 | **93,4** | **6,457422** |
| | 500 | 4509,6 | **710,7** | **6,344098** |
| Rastrigin | 5 | 144,5 | **102,8** | **1,400854** |
| | 10 | 207,2 | **133,8** | **1,546700** |
| | 50 | 435,9 | **76,6** | **5,675302** |
| | 500 | 3975,3 | **678,2** | **5,861422** |
| Sphere | 5 | **21,8** | 89,4 | 0,234987 |
| | 10 | **48,0** | 89,8 | 0,534789 |
| | 50 | **87,2** | 115,5 | 0,753521 |
| | 500 | 207,7 | **139,0** | **1,489253** |

**Table 2**: Speed test results

Tables 3 and 4 show the results obtained for multiple flocks approach.

| Function | | NP = 5 | |
|---|---|---|---|
| | | Standard | MF |
| Ackley | Best | 0 | 0 |
| | Mean | 0 | 0 |
| | Worst | 0 | 0 |
| | Stdev. | 0 | 0 |
| Griewank | Best | 2.0572E-12 | 0 |
| | Mean | 4.6408E-12 | 0 |
| | Worst | 5.9026E-12 | 0 |
| | Stdev. | 0 | 0 |
| Rastrigin | Best | 3.5786E-11 | 0 |
| | Mean | 5.2348E-11 | 0 |
| | Worst | 6.9383E-11 | 0 |
| | Stdev. | 0 | 0 |
| Sphere | Best | 0 | 0 |
| | Mean | 0 | 0 |
| | Worst | 0 | 0 |
| | Stdev. | 0 | 0 |

**Table 3**: Results for tests – 5 parameters

This method is trying to find better results than standard CS algorithm. Comparison between standard CS algorithm and multiple flocks approach is presented on different objective functions with various numbers of parameters. In order to make the comparison clearer, values below E-12 were assumed to be 0.

Results presented in Tables 3 and 4 show that multiple flocks approach can easily outperform standard CS algorithm, both in quality and consistency of results. Multiple flock modification of CS algorithm reached better results for every combination of benchmark function and dimensionality of search space.

| Function | | NP = 10 | | NP = 50 | |
|---|---|---|---|---|---|
| | | Standard | MF | Standard | MF |
| Ackley | Best | 3.5126E-12 | 0 | 2.6781E-11 | 0 |
| | Mean | 5.4467E-12 | 0 | 4.8050E-11 | 0 |
| | Worst | 6.9329E-12 | 0 | 1.5262E-10 | 0 |
| | Stdev | 0 | 0 | 3.3662E-12 | 0 |
| Griewank | Best | 2.8024E-11 | 0 | 1.5847E-10 | 0 |
| | Mean | 3.0716E-11 | 0 | 7.6320E-10 | 0 |
| | Worst | 5.5676E-11 | 0 | 3.7878E-9 | 0 |
| | Stdev | 0 | 0 | 6.5890E-11 | 0 |
| Rastrigin | Best | 2.0176E-10 | 0 | 1.5790E-9 | 0 |
| | Mean | 3.4511E-10 | 0 | 3.6750E-9 | 0 |
| | Worst | 5.6471E-10 | 0 | 5.6276E-9 | 0 |
| | Stdev | 8.5984E-11 | 0 | 1.6282E-10 | 0 |
| Sphere | Best | 3.2851E-12 | 0 | 3.9943E-11 | 0 |
| | Mean | 5.4185E-12 | 0 | 5.5326E-11 | 0 |
| | Worst | 6.1045E-12 | 0 | 7.5737E-11 | 0 |
| | Stdev | 0 | 0 | 0 | 0 |

**Table 4**: Result for tests – 10 and 50 parameters

# 4 Conclusion

In this paper we presented parallelized implementation of the CS algorithm for unconstrained optimization problems. The performance of this algorithm was investigated through the set of several experiments on well-known benchmark problems. The results obtained by this multithreaded algorithm are satisfying. It outperforms original, single threaded CS algorithm in both, speed and quality of results tests.

*References:*

[1] Nocedal J., Wright J. S., *Numerical Optimization,* Second Edition, Springer Berlin, 2006, p.669.

[2] Ólafsson S., *Metaheuristics,* Nelson and Henderson Handbook on Simulation, Handbooks in Operations Research and Management Science VII, Elsevier, 2006. pp. 633-654.

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

[4] Bonabeau E., Dorigo M., Theraulaz G., *Swarm intelligence: from natural to artificial systems*, Oxford University Press, New York, 1999, p. 307.

[5] Cecilia R., Tenreiro Machado J. A., *Crossing Genetic and Swarm Intelligence Algorithms to Generate Logic Circuits*, WSEAS Transactions on computers, Vol. 8, Issue 9, 2009, pp. 1419-1428.

[6] Kennedy J., Eberhart R., *Particle Swarm Optimization*, Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.

[7] Karaboga D., *An idea based on honey bee swarm for numerical optimization*, Technical Report TR06, Computer Engineering, Department, Erciyes University, Turkey, 2005.

[8] Karaboga D., Akay B., *A modified ABC for constrained optimization problems*, Applied soft computing, Vol. 11, Issue 3, 2011, pp. 3021-3031.

[9] Stanarevic N., Tuba M., Bacanin N., *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.

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

[11] Tuba M., Subotic M., Stanarevic N., *Modified cuckoo search algorithm for unconstrained optimization problems*, Proceedings of the European Computing Conference ECC'11, Paris, France, 2011, pp. 263-268

[12] Payne R. B., Sorenson M. D., Klitz K., *The Cuckoos*, Oxford University Press, 2005, p. 618.

[13] Pavlyukevich I. J., *Cooling down Lévy flights*, J. of Physics A: Mathematical and Theoretical, Vol. 40, No. 41, 2007, pp. 225-232.

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

[15] Sepehri N., Wan F.L.K, Lawrence P.D., Dumont G.A.M., *Hydraulic compliance identification using a parallel genetic algorithm*, Mechatronics, Vol. 4, Issue 6, 1994, pp. 617–633

[16] Pedemonte M., Nesmachnow S., Cancela H., *A survey of parallel ant colony optimization*, Applied Soft Computing, Vol. 11, Issue 8, 2011, pp. 5181-5197.

[17] Jovanovic R., Tuba M., Simian D., *Comparison of Different Topologies for Island-Based Multi-Colony Ant Algorithms for the Minimum Weight Vertex Cover Problem*, WSEAS Transactions on computers, Vol. 9, Issue 1, 2010, pp. 83-92.

[18] Li B., Wada K., *Communication latency tolerant parallel algorithm for particle swarm optimization*, Parallel Computing, Vol. 37, Issue 1, 2011, pp. 1-10.

[19] Subotic M., Tuba M., Stanarevic M, *Different approaches in parallelization of the artificial bee colony algorithm*, International Journal of mathematical models and methods in applied sciences, Vol. 5, Issue 4, 2011, pp. 755-762.