Swarm and Evolutionary Computation 29 (2016) 47–72

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

# Hybrid self-adaptive cuckoo search for global optimization

Uroš Mlakar, Iztok Fister Jr.*, Iztok Fister

*Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia*

ARTICLE INFO

*Article history:*
Received 11 August 2015
Received in revised form
3 March 2016
Accepted 14 March 2016
Available online 29 March 2016

*Keywords:*
Cuckoo search
Global optimization
Self-adaptation
Hybridization

ABSTRACT

Adaptation and hybridization typically improve the performances of original algorithm. This paper proposes a novel hybrid self-adaptive cuckoo search algorithm, which extends the original cuckoo search by adding three features, i.e., a balancing of the exploration search strategies within the cuckoo search algorithm, a self-adaptation of cuckoo search control parameters and a linear population reduction. The algorithm was tested on 30 benchmark functions from the CEC-2014 test suite, giving promising results comparable to the algorithms, like the original differential evolution (DE) and original cuckoo search (CS), some powerful variants of modified cuckoo search (i.e., MOCS, CS-VSF) and self-adaptive differential evolution (i.e., jDE, SaDE), while overcoming the results of a winner of the CEC-2014 competition L-Shade remains a great challenge for the future.

## 1. Introduction

Nowadays, nature-inspired algorithms are applied for solving the hardest optimization problems in theory and practice. The main advantage of these algorithms is searching for solutions using the principle of 'trial-and-error'. Thus, the possible solutions are generated using the various variation operators and their quality estimated by appropriate fitness function after testing. Two inspirations have crucial impact on the development of nature-inspired algorithms, i.e., the Darwinian evolution, and the behavior of social living animals (e.g., flocks of birds and shoals of fish) and insects (e.g., bees and ant colonies). The former inspiration influenced the development of evolutionary algorithms (EAs) [1], while the latter on swarm intelligence (SI) based algorithms [2]. Typically, the nature-inspired algorithms are stochastic and maintain a population of solutions that have improved their qualities over the generations. The new solutions are generated by applying the appropriate variation operators, like crossover, mutation, and move. The first two are employed by EAs, while the latter by SI-based algorithms.

Normally, EAs are appropriate for solving the problems where no user experience and no problem-specific knowledge exist. Therefore, these algorithms have been applied to almost every domain of optimization, simulation and modeling. According to a representation of solutions, EAs are divided into the following types: genetic algorithms (GA) [3], genetic programming (GP) [4], evolution strategies (ES) [5] and differential evolution (DE) [6,7], and evolutionary programming (EP) [8].

The concept of swarm intelligence was used for the first time by Beni and Wang [9] by development of cellular robots. Today, these algorithms are applied in optimization, control of robots, routing in a new generation of mobile networks, and other domains, where robustness and flexibility are demanded. The more frequently used SI-based algorithms are ant colony optimization [10], particle swarm optimization (PSO) [11], artificial bee colony (ABC) [12], artificial immune systems (AIS) [13], firefly algorithm (FA) [14], and symbiotic organisms search (SOS) [15].

Cuckoo search (CS) belongs to a SI-based family of stochastic population-based algorithms. It was developed by Yang and Deb [16] and is inspired by the brood parasitism of some cuckoo species laying their eggs in the nests of other bird species. Each position of the nest in the CS algorithm represents a solution of the problem to be solved. The destiny of a particular nest in each generation can be twofold: a nest with high-quality eggs is preserved for the next generation, while the nest with the lower-quality eggs is abandoned and replaced by a completely new nest.

According to the literature, a lot of cuckoo search variants have been proposed since its introduction in 2009. Shortly after the introduction of basic cuckoo search, Yang and Deb [17] extended the basic CS to a multiobjective CS algorithm intended for solving design optimization problems. A modified cuckoo search was developed by Walton et al., where they added an information exchange between the top eggs. On the other hand, many binary cuckoo search variants have also been proposed [18–20]. Recently, a lot of new CS variants have been proposed by using the hybridization of other algorithms as for example [21–23]. Incorporating adaptation to CS may also become a hot topic for further development. For example, recent adaptative CS variants

* Corresponding author.
*E-mail addresses:* uros.mlakar@um.si (U. Mlakar), iztok.fister1@um.si (I. Fister Jr.), iztok.fister@um.si (I. Fister).

http://dx.doi.org/10.1016/j.swevo.2016.03.001
2210-6502/© 2016 Elsevier B.V. All rights reserved.

are published in [24–27]. Loosely speaking, cuckoo search is used in myriads of applications in engineering, the real-world, energy management, finance industries too. A design optimization of truss structures was done by Gandomi et al. [28]. Additionally, Fateen and Bonilla-Petriciolet also applied CS algorithm within the chemistry domain [29,30], while Vázquez [31] employed the CS for training the spiking neural models. Gálvez et al. in [32] applied the CS for weighted Bayesian energy functional optimization.

Stochastic population-based algorithms are general enough to be appropriate for solving all classes of problems with which humans are confronted today. Unfortunately, the performances of these algorithms are the same when compared with regard to solving all classes of problems according to the No Free Lunch Theorem (NFL) [33]. However, the performances of general stochastic population-based algorithms can usually be improved in two ways. On the one hand, different algorithm parameter settings have a crucial impact on the performances of these algorithms. That is, the parameters that are valid by starting a search process become inappropriate when the process comes to a matured phase and vice versa. Consequently, the parameters should be changed during the run. In this sense, adaptation and self-adaptation [1] have been proposed within the computational intelligence community. The adaptation means that the parameters are changed according to a feedback from the search process, while the control parameters are saved into a representation of solutions and undergo acting the variation operators by self-adaptation. On the other hand, the stochastic population-based algorithms suffer from a lack of problem-specific knowledge that can generally be conducted by the so-called hybrid algorithms in forms of strategies/features, operators, construction and local search heuristics [34].

This paper proposes the hybrid self-adaptive CS algorithm (HSA-CS) using explicit control of exploration search strategies within the CS algorithm, self-adaptation of the CS control parameters and a population reduction feature. In the proposed HSA-CS, a search space is explored using three exploration search strategies, i.e., random long-distance search strategy, stochastic moderate-distance search strategy and stochastic short-distance search strategy [35]. While the first search strategy is devoted for exploring new solutions, the other two stochastic strategies are dedicated for exploiting the current solutions, where the search process is more directed towards the vicinity of the existing solutions [36]. Thus, the effects of both stochastic strategies are balanced using a special balancing parameter. The exploration/exploitation components of the CS search process are balanced by the control parameter setting. The self-adaptive control of parameter setting was employed by the HSA-CS. The motivation behind the population reduction [37] lies in the fact that the diversity of the population is high at the beginning and therefore more population members are needed. When the search process directs itself towards the more promising regions of the search space, the diversity of population decreases and the search process can reduce the population size accordingly.

The CEC-2014 benchmark function suite customized for global optimization was applied as a test-bed for assessing the quality of the HSA-CS. Recently, few CS algorithms have been developed for solving this problem suite due to the high complexity. As far as we know, Wang et al. [38] have been the only one's who have applied the CS to the similar problem suite. In this study, the results of the HSA-CS were compared with the other more promising variants of CS, like MOCS [39] and CS-VSF [38], the more powerful variants of DE, like jDE [40] and SaDE [41], and two under the first five qualified algorithms on the CEC-2014 competition, i.e., L-Shade [42] and MVMO [43]. Additionally, the original CS [16] and DE [6] have also been included in this comparative study. The study indicated the big potential for solving the global optimization problems of the proposed HSA-CS algorithm in the future.

The structure in the remainder of this paper is as follows. Section 2 describes the original CS algorithm. Section 3 discusses the proposed HSA-CS algorithm. Section 4 illustrates the experiments and results, while the paper is concluded with Section 5, where the performed work is summarized and directions for further development are outlined.

## 2. Cuckoo search algorithm

Cuckoo search (CS) is a contemporary stochastic nature-inspired population-based algorithm which was developed by Yang and Deb in late 2009 [16]. CS belongs to the SI-based algorithm family [44] that is inspired by the natural behavior of some cuckoo species along with their obligate brood parasitism. About this phenomenon, Payne in his book "The Cuckoos" [45] said:

"The parasitic breeding behavior of cuckoos has fascinated people for centuries. The brood-parasitic cuckoos lay their eggs in the nests of other kinds of birds, and never rear their own young."

The cuckoo egg hatches earlier than the host's and the cuckoo young's develop faster. Thus, the cuckoo young evict the eggs of the host species. There are several strategies how to lay eggs into a host nest by the female cuckoo. On the one hand, female cuckoos have fast laying behavior. On the other hand, some parasitic cuckoo females are specialized in laying eggs that resemble the eggs of the host species. Consequently, the host birds cannot distinguish the cuckoo eggs from their own and raise these as if they were their own.

Interestingly, not all cuckoo species demonstrate this brood-parasitism. For instance, the anis and the Guire cuckoos are cooperative breeders, where several cuckoo pairs sharing the same nest lay their eggs together and also care for their own young. Some cuckoo species live in solitaire pairs, build own nests and rear their young's.

However, the CS algorithm mimics the brood-parasitism behavior of cuckoo species. To trap the behavior of cuckoos in nature and adapt it to be suitable for using as a computer algorithm, authors have idealized three rules [16]:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest.
- The best nests with high-quality eggs will be carried over to the next generations.
- The number of available host nests is fixed and any egg laid by a cuckoo may be discovered by the host bird with a probability $p_a \in (0, 1)$. In this case, the host bird can either get rid of the egg, or simply abandon the nest and build a completely new one.

**Algorithm 1.** Original cuckoo search algorithm.

```
Input: Population of nests $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,D})^T$ for $i = 1 \ldots NP$, $MAX\_FE$.
Output: The best solution $\mathbf{x}_{best}$ and its corresponding value $f_{min} = \min(f(\mathbf{x}))$.
1: generate_initial_host_nest_locations;
2: $FE = 0$;
3: while termination_condition_not_meet do
4:   for $i = 1$ to $NP$ do
5:     $\mathbf{u}_i$ = generate_new_solution($\mathbf{x}_i$);
6:     $f_{trial}$ = evaluate_the_new_solution($\mathbf{u}_i$);
7:     $FE = FE + 1$;
8:     $j = \lfloor \text{rand}(0, 1) * NP + 1 \rfloor$;
9:     if $f_{trial} < f_j$ then
10:      $\mathbf{x}_j = \mathbf{u}_i$; $f_j = f_{trial}$;//replace the $j$-th random selected solution
11:    end if
```