Parameter Evolution Self-Adaptive Strategy and its Application for Cuckoo Search

Yifan He[1], Claus Aranha[2], and Tetsuya Sakurai[2]

[1] Graduate School of Systems and Information Engineering, University of Tsukuba
he.yifan.xs@alumni.tsukuba.ac.jp
[2] Faculty of Engineering, Information and Systems, University of Tsukuba
{caranha,sakurai}@cs.tsukuba.ac.jp

Abstract. Cuckoo Search (CS) is a simple yet efficient swarm intelligence algorithm based on Lévy Flight. However, its performance can depend heavily on the parameter settings. Though many studies have designed control strategies for scaling factor α, few have considered the adaption of the stability parameter β (of Lévy Flight). In this paper, we propose the Parameter Evolution Self-Adaptive strategy (PESA) to control β. PESA uses an evolutionary algorithm that runs in parallel to CS. We show that PESA can also be extended to control the parameters of other meta-heuristics, using Differential Evolution (DE) as a second example. We compare our strategy with the well-established self-adaptive strategy used in JADE, both in CS and DE, on classical benchmark functions. We discuss the increased flexibility of PESA and analyze the effect of changing the frequency of updating parameter values in CS.

Keywords: Self-adaption · Cuckoo Search · Lévy Flight.

1 Introduction

Cuckoo Search (CS) is a simple yet efficient meta-heuristic. The efficiency of CS derives mainly from the utilization of Lévy Flight (LF). LF is a type of random walk where the step length is drawn from a heavy-tailed distribution. Therefore, the parameters (i.e., scaling factor α and stability parameter β) of the step-length distribution can heavily influence CS performance. Although it has been suggested that a fixed value for the stability parameter ($\beta = 1.5$) could work on any problem, results from practical studies (for example, Kordestani's recent work [6]) shows that the optimal setting of β depends on the problem. Therefore, a (self-)adaptive strategy for controlling β is promising. However, while the (self-)adaptive strategies of other CS parameters such as α and p_a have been well-studied in the literature [8,11,16], the self-adaptive strategy for β has not been well discussed yet. Some studies have developed such a strategy for a limited range, or a set of discrete candidate values [6,11]. However, to the best of our knowledge, there is no self-adaptive strategy in the CS literature to control β in its complete domain (i.e., (0, 2]).

Self-adaptive strategies for parameter control have been well studied in other Evolutionary Algorithms. For example, JADE [20] has been designed to adapt

the two parameters of Differential Evolution (DE). In JADE, the algorithm continuously estimates the expectation value of good parameters based on successful parameters in the last generation. This algorithm has been shown to be effective and was further developed by other researchers in recent years [14]. However, the strategy in JADE cannot control β in CS without proper modification (as we found in our pre-experiment). So in this study, we propose a novel self-adaptive strategy that can control β in CS, as well as a variety of parameters in other meta-heuristics, called Parameter Evolution Self-Adaptive strategy (PESA).

PESA is an indicator-based strategy to control algorithm parameter values at an individual level. This strategy maintains two populations: a population of solutions and a population of algorithm parameters. PESA searches the parameter population using a secondary EA and a fitness function based on the progress of individuals in the solution population. PESA's structure makes it easy to apply it to several meta-heuristics and several types of parameters. This work is related to previous research on using EA to fine-tune control parameters. However, two key ideas of our method are the online evolution of parameters (selection and mutation of new parameters done at optimization time), and its general design that aims to work with several EAs.

In this work, we consider two implementations of PESA: First, we show an implementation to adaptively control the β parameter from CS in its full continuous interval (i.e., $[0.1, 1.9]$). Next, we consider an implementation to control the F and CR parameters from DE. We compare the PESA implementations for each search algorithm to JADE (and its corresponding modifications for use with the CS: JACS). On a set of 14 benchmark functions on 30 dimensions, PESA performs a better control of β when compared to JACS, while it has comparable results to JADE on the control of F and CR. This may indicate that the proposed strategy can be used with little modification for the control of parameters in a wide range of meta-heuristics. We also analyze the sensitivity of PESA to the frequency of control parameter updates.

2 Background

2.1 Cuckoo Search and Lévy Flight

CS was proposed by Yang in 2009 [18]. It has a simple structure, and many subsequent studies have shown it is an efficient meta-heuristic search. Its efficiency is derived from the powerful Lévy Flight operator (LF). Generally, LF is a random walk based on a stable distribution. Since a (non-Gaussian) stable distribution holds infinite variance, a large step of any arbitrary size can occur. This property helps CS escape from local optima during the optimization. The LF operator is formulated in Eq.(1), where $x^{(t)}$ is the position of search point x at time t. $L(\alpha, \beta)$ is a step generated from a stable distribution with scaling factor α and stability parameter β.

$$x^{(t)} = x^{(t-1)} + L(\alpha, \beta) \tag{1}$$

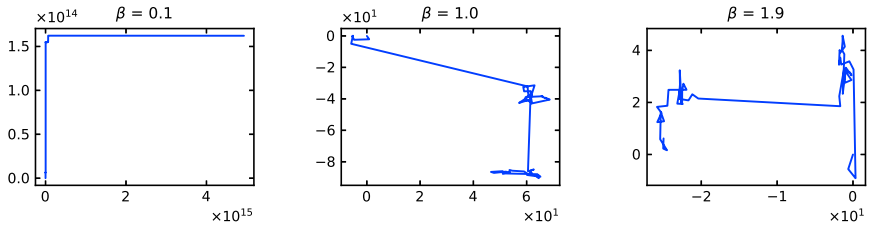


Fig. 1. 50-step Lévy Flights with different β and $\alpha = 1$ started from $(0, 0)$. When $\beta = 0.1$, the search range is nearly infinite, which shows a possibility of using Lévy Flight to search an unbounded space.

Fig. 1 illustrates 50 steps of three LFs with same $\alpha = 1$, same start point $(0, 0)$, but different βs. The scale of search ranges are quite different. Generally, a stable distribution holds infinite variance for $1 \leq \beta < 2$, and its expectation value is diverged when $0 < \beta < 1$. When $\beta = 2$, it becomes a Gaussian distribution. These properties show that any large step can be found in LF when $\beta \neq 2$, and LF can lose its average position and go everywhere in the search space when $0 < \beta < 1$. Therefore, LF with a proper β can be used for searching an unbounded space [4].

2.2 Parameter Adaption Strategies in Cuckoo Search

Because of the importance of parameter tuning, self-adaptive strategies are frequently studied and applied in meta-heuristics literature. In the case of CS, the adaption of the scaling factor α is a well-studied problem. Several studies have proposed adaptive strategies that adjust α based on generations [16,21] and individual fitness [12]. Researchers have also applied adaptive CS to solve some application problems [15,17].

On the other hand, the adaptation of the stability parameter β has not been studied as much as α in the adaptive CS literature. Mlakar has proposed a hybrid self-adaptive CS [11]. However, in his study, β is limited to [1.2, 1.8]. Kordestani has utilized multiple β values and designed learning automation to switch between fixed values (0.75 and 1.90) based on probability [6]. Abedi has proposed a similar design, where the solution is explored with β =1.0, 1.5, and 2.0 [1]. Lee has injected LF mutation into Evolutionary programming [7]. He has also developed an "adaptive" strategy, where four candidates are generated with fixed β (1.0, 1.3, 1.7, 2.0), and the best one is selected as offspring.

We find that most of the (self-)adaptive strategies in the CS literature are dealing with α and p_a. In a few existing studies for adaption of β, the candidate values are discrete [1,6,7] or a restricted range [11]. What is more, even in the case of deterministic parameter CS, performance with small β is seldom discussed.

2.3 Parameter Adaption in Other Meta-Heuristics

Since the (self)-adaption design of parameter β in CS has not been well discussed, we introduce self-adaptive adaption in another famous meta-heuristics named differential evolution (DE). Specifically, we introduce JADE [20]. Many variants of JADE and other (self-)adaptive design of DE can be found in the review work by Al-Dabbagh [2]. The full description of JADE can be referred to Zhang's study [20]. In JADE, each solution has an associated parameter value, and the selection of new parameters happens at the same time as the selection of new solutions. Parameters are obtained from a probability distribution, and the shape of this distribution is estimated during the optimization process.

Another idea for select parameter values of an EA is to use a secondary EA. The most well-known approach for this is the Meta-EAs [3,9]. Usually, a Meta-EA maintains multiple sub-populations of the primary EA, each with a corresponding parameter setting. The Meta-EA will analyze a full run of each sub-population, and therefore is computationally expensive.

A slightly different approach is to use a secondary EA that runs in parallel with the primary EA to guide parameter control in self-adaptive strategies. Posik has proposed a method to co-evolve the solutions and mutation steps in evolutionary strategy (ES) [13]. In this way, we can consider JADE as a parallel Estimation of Distribution Algorithm (EDA) guiding the DE's parameters. PLADE [19] has applied particle swarm optimization (PSO) to guide DE parameters. However, to our best knowledge, such a technique has not been applied to control the stability parameter in CS.

3 Proposed Method

3.1 Parameter Evolution Self-Adaptive Strategy

In this paper, we propose the Parameter Evolution Self-adaptive strategy (PESA), which is a generalization of JADE [20], PLADE [19], and Posik's study [13].

PESA uses two parallel populations of the same size: a *solution population* and a *parameter population*. Individual x_i in the solution population is the standard solution candidate for an EA, while p_i is the set of parameters that will be used to operate on x_i, and will be evaluated by a secondary, and possibly different, EA.

The fitness y_i of an individual x_i is calculated as usual, using the problem's fitness function f. On the other hand, the fitness indicator I_i of parameter candidate p_i is calculated by an indicator function g, which evaluates the search progress of the corresponding solution in the last n_{step} generations. This implies that the secondary EA evaluates, selects, and modifies the parameter population less frequently than the primary EA, this difference being controlled by the n_{step} parameter. An important characteristic to keep in mind is that this secondary EA does not consume extra fitness evaluations. The assessment of parameters is based on the solution information that has already been computed in the first EA. The outline of PESA is described in Algorithm 1.

Algorithm 1 Parameter Evolved Self-Adaptive Strategy (PESA)

1: **Input:** solution population $X = \{x_1, \cdots, x_N\}$, parameter population $P = \{p_1, \cdots, p_N\}$, fitness function f, indicator function g;
2: **while** termination criteria is not satisfied **do**
3: increase generation counter t;
4: evaluate solution fitness: $y_i^t = f(x_i^t)$
5: calculate offspring solution x_i^{t+1} based on x_i^t and parameter p_i;
6: **if** t is multiple of n_{step} **then**
7: evaluate parameter fitness: $I_i = g(x_i^t, y_i^t, x_i^{t-1}, y_i^{t-1}, \cdots, x_i^{t-n_{step}}, y_i^{t-n_{step}})$
8: calculate next parameter p_i' based on I_i and secondary EA;

As shown in Fig. 2, compared with Meta-EAs [3,9], each parameter setting in PESA corresponds with an individual rather than a sub-population of base-level EA. This further lead to two difference in the evolutionary process. First, a parameter is assessed by the search progress of only one individual. Second, in Meta-EAs, the fitness of a parameter is computed based on an independent run of one sub-population. However, in our PESA, individuals do not run independently; the search progress of one solution may be partially contributed by another individual. This shows that the fitness of a parameter in PESA should be well-designed.

Posik has proposed a co-evolutionary algorithm for real parameter optimization [13]. In that study, the two populations, a population of solutions and a population of mutation steps, are co-evolved. Compared to his study, our method co-evolves solutions with parameters rather than mutation steps. An essential difference between parameters and mutation steps is that a parameter can generate multiple types (size) of step sizes. Therefore, evolving parameters is generally more difficult. To deal with this problem, we introduce a multi-generation assessment, where each parameter is assessed for n_{step} generations.

It is not hard to note that JADE (without external archive) [20] is a special case of PESA. PESA determines that the quality of parameters is assessed by an indicator function, and then selected independently from the selection process of solutions. In JADE, there is no explicit indicator function, and the selection of parameters and solutions are both based on the fitness of the solution. This could be expressed in PESA as $I_i = \max(y_i^{t-1} - y_i^t, 0)$ as the indicator function.

3.2 Parameter Evolution Cuckoo Search

PESA is a general strategy for controlling parameters of an EA, which requires the definition of the indicator function and the secondary EA rules for generating new parameters.

In this paper, we introduce Parameter Evolution Cuckoo Search (PECS) as a specific implementation of PESA for cuckoo search. Since parameter α and β are highly correlated, we design self-adaption for stability parameter β but tune α for simplicity. In PECS, the solutions are evolved by CS (in Line 5 of Algorithm 1), and the parameters are evolved by another EA (Line 8 of the same algorithm).

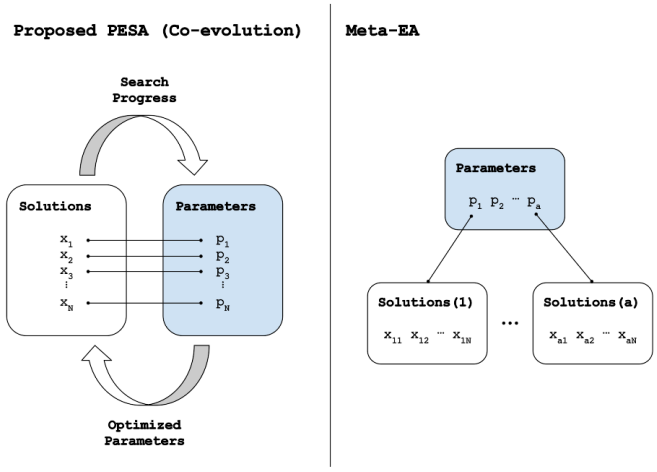


Fig. 2. Conceptual Difference between Parameter Evolved Self-Adaptive Strategy (PESA) and Meta-EA. Meta-EA evaluates a parameter set by running an entire subpopulation with that parameter, while PESA evaluates a parameter set based on a single corresponding individual.

The indicator function that expresses the fitness of a parameter set is computed cumulatively by the corresponding improvement over n_{step} generations of CS, as in Eq.(2). A new parameter set is generated following the crossover step described in Eq.(3) and the mutation step in Eq.(4), with probability p_c and p_m, respectively. In these equations, β_{select} is a stability parameter chosen by roulette selection, every time before the parameter reproduction step. The selection probability of β_i is equal to $I_i/\Sigma I_i$. σ is a random uniform value between 0 and 1. The selection between the new parameter set and the old one is based on the comparison of a pair of positions, before and after implementing LF (i.e., x_i and x_i'). This marks an important difference between the selection of solutions by CS and the selection of parameters by the secondary EA.

$$I_i += \max\{f(x_i) - f(x_i'), 0\} \tag{2}$$

$$\beta_i' = \beta_i + \sigma * (\beta_{select} - \beta_i),\ \sigma \in [0, 1] \tag{3}$$

$$\beta_i'' = \beta_i' + L(\theta, \lambda) \tag{4}$$

Implementation of Lévy Flight. To implement LF, we use Eq.(1) and Eq.(5). The first case of the equation below is based on Gutowski's study [4], where a simulation method is constructed to generate only positive LF random numbers for $\beta \in (0, 2)$. We multiply {-1,1}, which means the equal probability of 1 and -1, to achieve both positive and negative LF random numbers. The symbol *rand*

stands for a uniform random number in [0, 1]. The second case is implemented by Mantegna's algorithm [10]. This algorithm approximates a symmetrical stable distribution, however, for a limited range of $\beta \in [0.3, 1.99]$.

$$L(\alpha, \beta) = \begin{cases} \alpha \cdot \{-1, 1\} \cdot (rand^{-\frac{1}{\beta}} - 1), & 0.1 \leq \beta < 0.3 \\ \alpha \cdot \frac{u}{|v|^{1/\beta}} & 0.3 \leq \beta \leq 1.9 \end{cases} \tag{5}$$

4 Experiments

We show our PESA can control the stability parameter β (with a tuned α) in CS, as well as F and CR in DE. In other words, we compare the performance of different adaptive strategies to solve the parameter space of CS and DE. The source code for all algorithms, extra figures and data are in our public website [3].

4.1 Comparison with Self-Adaptive Strategy in JADE

CS-based Algorithms Totally, we have four algorithms to compare, namely PECS, JACS, Random Parameter CS (RPCS), and CS (Tuned).

- **PECS:** CS where β is controlled by the proposed PESA.
- **JACS:** CS where β is controlled by JASA. In our pre-experiments, we find that without a proper modification, JACS performs poorly on most of the problems. Thus, we implemented a modification to realize the quality of LF in terms of fitness improvement by applying weighted mean. This modification is similar to what proposed by Tanabe [14].
- **RPCS:** CS where β is randomly generated from [0.1, 1.9] before mutation.
- **CS (Tuned):** CS where β is fine-tuned on each benchmark problem.

DE-based Algorithms To test PESA's performance on controlling DE, we include PEDE (DE controlled by PESA). As comparison methods, we use JADE and Random Parameter DE (RPDE). The mutation method of all three methods is DE/current-to-pbest/1, and all three algorithms (PEDE, JADE, RPDE) are implemented without an external archive for simplicity.

Experimental Settings for CS-based Algorithms For a fair comparison, we tuned parameters of all CS-based algorithms based on the average evaluation cost and average fitness of 21 runs with maximum evaluations of 15,000 on Sphere and Rastrigin. We set the population size N as 20, which is a common value. We tuned α and β in pairs to get a proper α. Based on the results, we set α on all problems as 1e-07. We then tuned p_a to 0.1. For JACS, we tuned the scaling factor of Cauchy distribution γ=0.1 and learning rate c=0.1. We set the initial value of μ_β as 1.0 (average of 0.1 and 1.9). For PECS, we set θ and λ to

[3] https://y1fanhe.github.io/research/bioma2020

0.1 and 1.0, respectively. This setting is the same as LF by Cauchy distribution with a scaling factor of 0.1. We tuned p_c and p_m to 0.7 and 0.3, respectively. We set n_{step} as 5, and discuss the influence of this parameter in Section 4.2. For CS (Tuned), we run algorithm with β={ 0.1, 0.2, ..., 1.9 } and chose the best β to compare.

In our experiment, a total of 14 benchmark problems (30-dimension) are included. We have implemented them based on Jamil's review work on continuous benchmark problems [5]. They are F_1: Sphere, F_2: Sum Squares, F_3: Rosenbrock, F_4: Zakharov, F_5: Ackley, F_6: Alpine N.1, F_7: Periodic, F_8: Styblinski-Tank, F_9: Rastrigin, F_{10}: Griewank, F_{11}: Schwefel, F_{12}: Salomon, F_{13}: Xin-She Yang's N.2, and F_{14}: Xin-She Yang's N.4 function. F_1 - F_4 are unimodal problems and F_5 - F_{14} are multimodal problems.

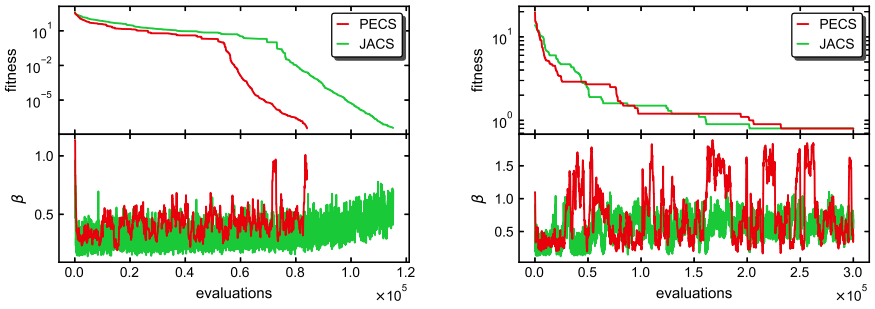
The experiment is performed with 31 repetitions. Each algorithm will run with a maximum evaluation of 300,000. The termination criterion is when the fitness meets the tolerance (to the optimal fitness value). The tolerance for each problem is computed as follows. We first run CS with β from 0.1 to 1.9 for 300,000 evaluations for 31 repetition and use the distance from the best average fitness to the optimal fitness as tolerance. We record the success rate in 31 runs as well as the mean and standard deviation of the number of evaluation numbers in successful runs as results.

Experimental Settings for DE-based Algorithms We use a population of 100 and a pbest rate of 0.05 for all methods. F and CR are controlled by a self-adaptive strategy within [0, 1]. For JADE, the parameters used for self-adaptive strategy is μ_F=μ_{CR}=0.5, γ=0.1, c=0.1. These settings are the same as in Zhang's study [20]. We use the tuned parameters in the previous experiments for the parameters of the adaptive strategy in PEDE. The maximum evaluations are set to 300,000. We run JADE in the same procedure to set tolerance for DE-based algorithms.

Experimental Results Table 1 and Table 2 presents the results of six CS-based algorithm and three DE-based algorithms on F_1 - F_{14}, respectively. We additionally performed the aggregated Friedman test and the aggregated Wilcoxon Signed-Rank test. The results have shown that the improvement of PECS over JACS and RPCS in terms of the number of evaluations is statistically significant (p-values 0.028 and 0.003, respectively), while there is no statistical difference between all four CS-based methods in terms of success rate (p-value=0.155). JADE is better than PEDE in terms of success rate, however, without statistical significance (p-value=0.058). All DE-based algorithms perform in the same tier in terms of the number of evaluations (p-value=0.054). These results show that PESA can work better than JASA on controlling CS and perform a comparable result with JASA on controlling DE. This may indicate that our PESA can work in a more general case compared to JASA.

Table 1. Success rate (SR), mean and standard deviation of evaluation numbers for CS-based algorithms (The numbers after CS are the best values of β)

Fun.	Method	SR	Mean	Std.	Fun.	Method	SR	Mean	Std.
F_1	PECS	1.00	**5.47e+04**	6.67e+03	F_8	PECS	1.00	**7.23e+04**	2.79e+04
	JACS	1.00	9.03e+04	2.92e+03		JACS	1.00	1.00e+05	1.09e+04
	RPCS	1.00	7.07e+04	3.89e+03		RPCS	1.00	1.74e+05	3.95e+04
	CS (0.6)	0.68	2.63e+05	2.44e+04		CS (0.4)	0.65	2.62e+05	2.11e+04
F_2	PECS	1.00	**5.80e+04**	5.70e+03	F_9	PECS	**0.94**	1.73e+05	4.89e+04
	JACS	1.00	9.34e+04	3.47e+03		JACS	0.81	1.57e+05	3.86e+04
	RPCS	1.00	7.90e+04	4.88e+03		RPCS	0.16	2.66e+05	2.86e+04
	CS (0.5)	0.94	2.22e+05	2.98e+04		CS (0.4)	0.81	2.50e+05	3.17e+04
F_3	PECS	0.81	8.61e+04	7.13e+04	F_{10}	PECS	0.71	4.37e+04	3.92e+04
	JACS	0.61	8.76e+04	5.67e+04		JACS	**0.81**	5.62e+04	1.59e+04
	RPCS	0.52	1.24e+05	6.37e+04		RPCS	0.26	1.27e+05	7.47e+04
	CS (0.4)	**0.87**	7.41e+04	5.34e+04		CS (0.2)	0.77	3.01e+04	4.49e+03
F_4	PECS	0.00	-	-	F_{11}	PECS	0.71	2.29e+05	4.40e+04
	JACS	0.00	-	-		JACS	0.39	2.29e+05	3.11e+04
	RPCS	0.00	-	-		RPCS	0.00	-	-
	CS (0.3)	**0.77**	2.71e+05	1.86e+04		CS (0.2)	**0.94**	2.30e+05	3.01e+04
F_5	PECS	1.00	**6.15e+04**	5.96e+03	F_{12}	PECS	0.00	-	-
	JACS	1.00	1.03e+05	4.57e+03		JACS	0.00	-	-
	RPCS	1.00	1.14e+05	1.19e+04		RPCS	0.00	-	-
	CS (0.5)	0.68	2.73e+05	1.48e+04		CS (0.2)	**0.61**	1.31e+05	5.45e+04
F_6	PECS	1.00	3.91e+04	2.68e+04	F_{13}	PECS	1.00	1.76e+05	3.31e+04
	JACS	1.00	**2.89e+04**	2.13e+04		JACS	1.00	2.16e+05	3.44e+04
	RPCS	0.97	2.79e+04	1.64e+04		RPCS	0.00	-	-
	CS (0.3)	0.74	1.38e+05	6.16e+04		CS (0.3)	1.00	**5.87e+04**	1.37e+04
F_7	PECS	1.00	**1.54e+04**	2.45e+03	F_{14}	PECS	1.00	**2.61e+04**	2.52e+03
	JACS	1.00	1.75e+04	1.74e+03		JACS	1.00	4.08e+04	2.67e+03
	RPCS	1.00	1.60e+04	1.90e+03		RPCS	1.00	4.83e+04	3.54e+03
	CS (0.6)	1.00	5.83e+04	1.14e+04		CS (0.5)	1.00	1.31e+05	1.88e+04



(a) F_9 **(b)** F_{12}

Fig. 3. Mean of β and fitness in best run on F_9 and F_{12} for adaptive CS

Table 2. Success rate (SR), mean and standard deviation of evaluation numbers for DE-based algorithms

Fun.	Method	SR	Mean	Std.	Fun.	Method	SR	Mean	Std.
	PEDE	1.00	3.43e+04	2.24e+03		PEDE	0.48	6.38e+04	7.40e+03
F_1	JADE	1.00	**3.33e+04**	1.20e+03	F_8	JADE	**0.90**	8.96e+04	6.10e+03
	RPDE	1.00	3.70e+04	7.28e+02		RPDE	0.35	6.79e+04	1.03e+04
	PEDE	1.00	3.94e+04	2.13e+03		PEDE	1.00	**1.52e+05**	5.24e+03
F_2	JADE	1.00	**3.78e+04**	1.33e+03	F_9	JADE	1.00	1.77e+05	3.20e+03
	RPDE	1.00	4.21e+04	7.51e+02		RPDE	0.00	-	-
	PEDE	0.03	2.75e+05	0.00e+00		PEDE	0.74	5.38e+04	6.56e+03
F_3	JADE	**0.55**	1.99e+05	7.28e+03	F_{10}	JADE	**1.00**	5.57e+04	1.62e+04
	RPDE	0.00	-	-		RPDE	0.87	5.66e+04	4.76e+03
	PEDE	**1.00**	9.92e+04	5.69e+03		PEDE	**0.32**	2.10e+05	4.58e+04
F_4	JADE	0.94	1.05e+05	1.20e+04	F_{11}	JADE	0.16	2.49e+05	4.18e+04
	RPDE	1.00	1.66e+05	4.42e+03		RPDE	0.00	-	-
	PEDE	1.00	**5.76e+04**	1.98e+03		PEDE	0.00	-	-
F_5	JADE	1.00	6.47e+04	2.64e+03	F_{12}	JADE	**0.16**	6.87e+04	1.08e+04
	RPDE	1.00	7.34e+04	1.08e+03		RPDE	0.03	1.41e+05	0.00e+00
	PEDE	1.00	**2.38e+04**	1.20e+04		PEDE	0.65	1.86e+05	4.65e+04
F_6	JADE	1.00	3.80e+04	1.16e+03	F_{13}	JADE	**0.71**	1.29e+05	2.76e+04
	RPDE	1.00	5.94e+04	3.36e+03		RPDE	0.00	-	-
	PEDE	1.00	9.86e+04	8.02e+03		PEDE	0.10	2.42e+05	2.43e+04
F_7	JADE	1.00	**9.79e+04**	6.03e+03	F_{14}	JADE	**1.00**	2.67e+05	1.31e+04
	RPDE	0.00	-	-		RPDE	0.97	1.24e+05	4.55e+04

Discussion It is interesting to see how β evolves with different strategies. As an example, the evolutionary process best fitness and mean of β in the best runs of PECS and JACS on F_9 and F_{12} are plotted in Fig. 3. On F_9, we can find that both methods decrease β at first to perform a global search, and increase to perform precious local search later. What is more, our PECS increases β much earlier than JACS, and thus holds a better convergence speed. On F_{12}: Salomon function, the evolutionary process of β in PECS is periodic. The ring-shaped local optima occur periodically on the domain. This may indicate that the search process of parameters is too greedy, and the parameters are evolved to large β at an early phase. On this problem, the performance of all self-adaptive CS is worse than a fine-tuned CS with β=0.2. This also shows the nature of self-adaptive EAs; the strategy can only reward the short-term benefits.

4.2 Comparison on Using Different n_{step}

In our PECS, a parameter is assessed with n_{step} generations. The setting of n_{step} can influence performance. A small n_{step} leads to an inaccurate assessment on the performance of parameters, while with a large n_{step}, parameters have less chance to be updated. Also, the solution may move far away from the current position after a large generation. Fig. 4 presents the evolutionary process of β

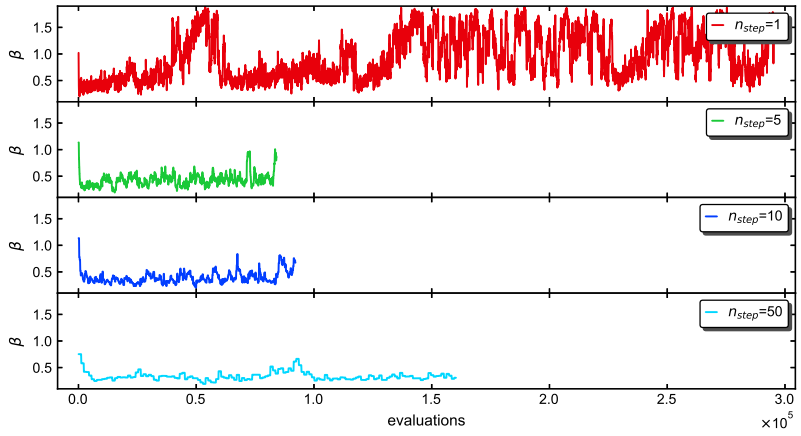


Fig. 4. Mean of β in best run of PECS on F_9 with different n_{step}

by n_{step}={ 1, 5, 10, 50 } on F_9. It is not hard to find that on all the three problems, small n_{step} leads to a rapid change in β. On F_9, when n_{step}=1, the fitness keeps a high value for a large number of evaluations, which indicates that the algorithm cannot escape from the local optimal. However, when n_{step}=50, the convergence speed is slower than n_{step}=5 and 10. A similar observation can be found in most of the testing problems. Therefore, n_{step}=5 or 10 should be a good choice for this parameter.

5 Conclusions

In this study, we proposed a novel self-adaptive strategy (PESA) to control the stability parameter of LF in CS from a wide range ([0.1, 1.9]). The proposed self-adaptive strategy is a co-evolution between solutions and parameters. We also showed that the proposed PESA is a generalization of many literature methods, such as JADE [20]. In the experiments, we showed that for both CS and DE, the proposed PESA could perform better or at least comparable results, compared with the strategy in JADE (JASA).

In the future, we will assess our methods on CEC benchmarks and compared them with other state-of-the-art algorithms. We will apply our strategy to more EAs to test its flexibility. We will extend our method to control multiple parameters simultaneously. Also, it is essential to discover a new metric to guide the search for parameters. What is more, it is also interesting to adapt our method to a multi-objective evolutionary algorithm.

References

1. Abedi Firouzjaee, H., Kordestani, J.K., Meybodi, M.R.: Cuckoo search with composite flight operator for numerical optimization problems and its application in tunnelling. Engineering Optimization **49**(4), 597–616 (2017)
2. Al-Dabbagh, R.D., Neri, F., Idris, N., Baba, M.S.: Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy. Swarm and Evolutionary Computation **43**, 284–311 (2018)
3. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on systems, man, and cybernetics **16**(1), 122–128 (1986)
4. Gutowski, M.: Lévy flights as an underlying mechanism for global optimization algorithms. arXiv preprint math-ph/0106003 (2001)
5. Jamil, M., Yang, X.S.: A literature survey of benchmark functions for global optimisation problems. International Journal of Mathematical Modelling and Numerical Optimisation **4**(2), 150–194 (2013)
6. Kordestani, J.K., Firouzjaee, H.A., Meybodi, M.R.: An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems. Applied Intelligence **48**(1), 97–117 (2018)
7. Lee, C.Y., Yao, X.: Evolutionary programming using mutations based on the lévy probability distribution. IEEE Transactions on Evolutionary Computation **8**(1), 1–13 (2004)
8. Li, X., Yin, M.: Modified cuckoo search algorithm with self adaptive parameter method. Information Sciences **298**, 80–97 (2015)
9. Luke, S., Talukder, A.K.A.: Is the meta-ea a viable optimization method? In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 1533–1540 (2013)
10. Mantegna, R.N., Stanley, H.E.: Stochastic process with ultraslow convergence to a gaussian: the truncated lévy flight. Physical Review Letters **73**(22), 2946 (1994)
11. Mlakar, U., Fister Jr, I., Fister, I.: Hybrid self-adaptive cuckoo search for global optimization. Swarm and Evolutionary Computation **29**, 47–72 (2016)
12. Ong, P.: Adaptive cuckoo search algorithm for unconstrained optimization. The Scientific World Journal **2014** (2014)
13. Posik, P.: Real-parameter optimization using the mutation step co-evolution. In: 2005 IEEE Congress on Evolutionary Computation. vol. 1, pp. 872–879. IEEE (2005)
14. Tanabe, R., Fukunaga, A.: Success-history based parameter adaptation for differential evolution. In: 2013 IEEE congress on evolutionary computation. pp. 71–78. IEEE (2013)
15. Valian, E., Tavakoli, S., Mohanna, S., Haghi, A.: Improved cuckoo search for reliability optimization problems. Computers & Industrial Engineering **64**(1), 459–468 (2013)
16. Walton, S., Hassan, O., Morgan, K., Brown, M.: Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solitons & Fractals **44**(9), 710–718 (2011)
17. Wang, J., Zhou, B.: A hybrid adaptive cuckoo search optimization algorithm for the problem of chaotic systems parameter estimation. Neural Computing and Applications **27**(6), 1511–1517 (2016)
18. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: 2009 World congress on nature & biologically inspired computing (NaBIC). pp. 210–214. IEEE (2009)

19. Zhan, Z.H., Zhang, J.: Self-adaptive differential evolution based on pso learning strategy. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 39–46 (2010)
20. Zhang, J., Sanderson, A.C.: Jade: adaptive differential evolution with optional external archive. IEEE Transactions on evolutionary computation **13**(5), 945–958 (2009)
21. Zhang, Y., Wang, L., Wu, Q.: Modified adaptive cuckoo search (macs) algorithm and formal description for global optimisation. International Journal of Computer Applicationsin Technology **44**(2), 73 (2012)