Kang Huang, Yongquan Zhou*, Xiuli Wu and Qifang Luo

A Cuckoo Search Algorithm With Elite Opposition-Based Strategy

DOI 10.1515/jisys-2015-0041
Received April 30, 2015; previously published online October 13, 2015.

Abstract: In this paper, a cuckoo search (CS) algorithm using elite opposition-based strategy is proposed. The opposite solution of the elite individual in the population is generated by an opposition-based strategy in the proposed algorithm and form an opposite search space by constructing the opposite population that locates inside the dynamic search boundaries, then, the search space of the algorithm is guided to approximate the space in which the global optimum is included by simultaneously evaluating the current population and the opposite one. This approach is helpful to obtain a tradeoff between the exploration and exploitation ability of CS. In order to enhance the local searching ability, local neighborhood search strategy is also applied in this proposed algorithm. The experiments were conducted on 14 classic benchmark functions and 28 more complex functions from the IEEE CEC'2013 competition, and the experimental results, compared with five other meta-heuristic algorithms and four improved cuckoo search algorithms, show that the proposed algorithm is much better than the compared ones at not only the accuracy of solutions but also for the convergence speed.

Keywords: Cuckoo search algorithm, elite opposition-based strategy, diversity of population, accuracy of solutions.

1 Introduction

In recent years, more and more bio-inspired algorithms were proposed [12], such as, genetic algorithm (GA) [10], charged system search (CSS) method [18], particle swarm optimization (PSO) [19], firefly algorithm (FA) [36], differential evolution optimization (DE) [28], monkey search (MS) [23], fly optimization algorithm (FOA) [24], invasive weed optimization (IWO) [22], bat algorithms (BA) [37], colliding bodies optimization (CBO) [17], ray optimization (RO) [16], dolphin optimization [15], etc. Because of its advantages in global, parallel efficiency, robustness, and universality, swarm intelligence algorithms have been widely used in engineering optimization, scientific computing, automatic control, and other fields.

The cuckoo search (CS) is novel meta-heuristic algorithms, developed in 2009 by Yang and Deb [38–40]. The CS is based on the brood parasitism of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights [25], rather than by simple isotropic random walks. Recent studies show that CS is potentially far more efficient than PSO and genetic algorithms. Cuckoos are fascinating birds, not only because of the beautiful sounds they can make but also because of their 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. Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species).

***Corresponding author: Yongquan Zhou,** College of Information Science and Engineering, Guangxi University for Nationalities, Nanning 530006, China, e-mail: yongquanzhou@126.com; and Guangxi High School Key Laboratory of Complex System and Computational Intelligence, Nanning 530006, China
Kang Huang and Xiuli Wu: School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
Qifang Luo: College of Information Science and Engineering, Guangxi University for Nationalities, Nanning 530006, China; and Guangxi High School Key Laboratory of Complex System and Computational Intelligence, Nanning 530006, China

The CS algorithm has been applied in many areas of optimization. For example, in the engineering design applications, the CS has superior performance over other algorithms for a range of continuous optimization problems such as spring design, welded beam design, and design of steel frame problems [7, 8, 14, 39]. In addition, a modified cuckoo search by Walton et al. [32] has demonstrated to be very efficient in solving nonlinear problems such as mesh generation. Vazquez [31] used the CS to train spiking neural network models, while Chifu et al. [4] optimized semantic web service composition processes using the CS. Furthermore, Kumar and Chakarverty [20] achieved optimal design for reliable embedded system using the CS, and Kaveh and Bakhshpoori [13] used the CS to successfully design steel frames. Yildiz [41] has used the CS to select optimal machine parameters in milling operation with enhanced results, while Zheng and Zhou [42] provided a variant of the CS using the Gaussian process.

On the other hand, a discrete CS algorithm as been proposed by Tein and Ramli [30] to solve nurse scheduling problems. The CS has also been used to generate independent paths for software testing and test data generation [5, 26, 27]. As a further extension, Yang and Deb [40] produced a multi-objective CS for design engineering applications. For multi-objective scheduling problems, a significant progress was made by Chandrasekaran and Simon [3] using cuckoo search algorithm, which demonstrated the superiority of their proposed methodology.

Recently, many swarm intelligence algorithms with the elite information have been proposed, such as a reverse particle swarm optimization algorithm-based elite proposed by Zhou [43], Takahama proposed a differential evolution algorithm based on an elite set of feasible solutions for solving constrained optimization problems [29]. They use a wealth of information carried by the elite and effectively improve the convergence speed and accuracy.

In order to overcome the limited performance of standard CS on complex problems, improve the ability of global searching and local searching, a novel dynamic CS algorithm with elite opposition-based strategy (EOS) is proposed in this paper.

The rest of this paper is organized as follows. The basics of CS algorithm is presented in the next section, and the CS with EOS is presented in Section 3, and the concept of EOS [2], the local neighborhood search strategy (LNSS) [6], and dynamic proportion strategy (DPS) are explained, followed by the detailed discussion of the proposed EOSCS algorithm. The experiment results are illustrated in detail in Section 4, compared to CLSPSO [1], DE [28], BA [37], FOA [24], CS [38] and other improved CS algorithms(CCS [35], GCS [42], DECS [33], DDICS [34]). Finally, some remarks and conclusions are provided in Section 5.

2 Cuckoo Search Algorithm

The CS optimization technique was introduced by Yang and Deb recently [38]. Cuckoos have a belligerent reproduction tactic that involves the female laying her fertilized eggs in the nest of another species so that the surrogate parents unwittingly raise her brood. Sometimes the cuckoo's egg in the nest is revealed, and the surrogate parents throw it out or dump the nest and start their own brood elsewhere. The CS optimization algorithm considered various design parameters and constraints, the three main idealized rules on which it is based are as follows:

1. Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest;
2. The best nests with a high quality of eggs will carry over to the next generations;
3. The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest and build a completely new nest. Based on the above three rules, the basic process of CS algorithm can be summarized in the pseudo-code shown in Algorithm 1.

For simplicity, this last assumption can be approximated by the fraction p_a of n nests in the current iteration, which need to be replaced by new nests (with new random solutions) in the next iteration. In addition, each nest can represent a set of solutions; CS can thus be extended to the type of meta-population algorithm. The

Algorithm 1. Cuckoo search via Lévy flight algorithm (CS).

Begin
Objective function $f(x)$, $x = (x_1, x_2, \ldots, x_d)^T$
Generate initial population of n host nests $x_i = (i = 1, 2, \ldots, n)$
While ($t <$ Max Generation) or (stop criterion)
Get a cuckoo randomly by Lévy flights
Evaluate its quality/fitness F_i
Choose a nest among n (say, j) randomly
If ($F_i > F_j$),
replace j by the new solution;
End
A fraction (p_a) of worse nests are abandoned and new ones are built;
Keep the best solutions (or nests with quality solutions);
Rank the solutions and find the current best
End while
Post-process results and visualization
End

number of parameters to be tuned in the CS is less than other nature-inspired techniques. The technique has been demonstrated successfully on some benchmark functions and is found to be far better than other approaches including the advanced particle swarm optimization approach.

3 Cuckoo Search With Elite Opposition-Based Strategy Algorithm

In order to enhance the global searching and local searching abilities, we applied three optimization strategies to basic the CS algorithm; those were EOS [2], local neighborhood searching strategy (LNSS) [6], and dynamic proportion strategy (DPS).

3.1 Elite Opposition-Based Strategy (EOS)

In the CS algorithm, the Lévy flight can improve the diversity of the population and strengthen the global search ability of the algorithm. In contrast, if its step size is set too long or too short, it also can run out of the global optimal value. Thus, we add EOS to global search process to enhance its exploitation ability.

First, we should explain a model (elite opposition-Based model). Its main idea is to assess the current solution and reverse solution, choose a better solution as the next generation of individuals. We suppose that there exists a point $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ is a parameter vector, and its dimension is D, $x_1, x_2, \ldots, x_D \in R$, and $x_j \in [a_j, b_j]$. The opposition point $\breve{x}_i = (\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_D)$ is defined in the formula below:

$$\breve{x}_j = a_j + b_j - x_j \tag{1}$$

In this paper, we use an EOS to enhance the global search ability of cuckoo search algorithm. It calculated and evaluated the population reverse solution, and the elite opposition-based model can be expressed in the following formula:

$$\breve{x}^e_{i,j} = k \cdot (da_j + db_j) - x^e_{i,j} \tag{2}$$

Where $x^e_{i,j} \in [a_j, b_j]$, $k \in U(0, 1)$. k is a generalize factor. da_j and db_j are two dynamic boundary. The dynamic boundary can be expressed according to following formula:

$$da_j = \min(x_{i,j}),\ db_j = \max(x_{i,j}) \tag{3}$$

In order to save the search experience, we use dynamic boundary instead of fixed boundary, it can reduce the space of search. If the opposition solutions exceed the boundary, we use a randomly number (between 0 and 1) to reset it. The reset number can be expressed according to following formula:

$$\breve{x}_{i,j}^{e} = rand(a_j, b_j) \text{if } \breve{x}_{i,j}^{e} < a_j \text{ or } b_j > \breve{x}_{i,j}^{e} \tag{4}$$

EOS can produce a population of inverse solution and enhance the new search space. It can improve the diversity of population and strengthen the global search ability of the algorithm. What is more, by comparing the current solution with inverse solution, the best solution is selected as the elite into the next generation.

3.2 Local Neighborhood Search Strategy (LNSS)

The CS algorithm uses a random step change to do local search. The experiment results show that the local search ability is limited. Thus, we add LNSS to the local search process to enhance its exploitation ability.

In the local neighborhood search model, each vector uses the best vector of only a small neighborhood rather than the entire population to do the location update operation. We suppose that there exists a differential population $P_G = [X_{1,G}, X_{2,G}, \ldots, X_{i+1,G}]$, and each $X_{i,G}(i = 1, 2, \ldots, NP)$ is a parameter vector, and its dimension is D. Each vector subscript indices are randomly divided to ensure the diversity of each neighborhood. To each vector $X_{i,G}$, we can define a neighborhood, and the radius is $k(2k+1 < NP)$. The neighborhood consists of vector $X_{i-k,G}, \ldots, X_{i,G}, \ldots, X_{i+k,G}$. Assuming that the vectors accord the subscript indices in a ring topology structure. We can take $X_{NP,G}$ and $X_{2,G}$ as two direct neighbors of $X_{1,G}$. The concept of the local neighborhood model is shown in Figure 1. The neighborhood topology here is static and determined by the collection of vector subscript indices. The local neighborhood model can be expressed in the following formula:

$$X_{I,G+1} = X_{i,G} + \alpha(X_{l\_best,G} - X_{i,G}) + \beta(X_{p,G} - X_{q,G}) \tag{5}$$

where $X_{l\_best,G}$ is the best vector of $X_{i,G}$ neighborhood, and $p, q \in [i-k, i+k](p \neq q \neq i)$, α, β are two scale factors.

3.3 Dynamic Proportion Strategy (DPS)

In the CS algorithm, the local search and global search proportion is controlled by a fixed parameter Pa. We suppose that a reasonable algorithm should do more global search at the beginning of the searching process and the global search should be less in the end. Thus, we applied the DPS to adjust the proportion of two kinds of searching process. The value of proportion Pa can alter according to following formula:

$$Pa = Pa - 0.1 \cdot \frac{(iterMax - t)}{iterMax} \tag{6}$$

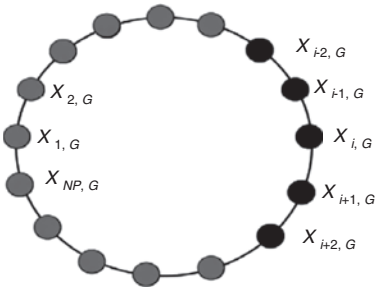

Figure 1: Neighborhood Ring Topology.

where *iterMax* is the maximum iterations of the EOSCS, and t is the current iteration. Specific implementation steps of EOSCS with elite opposition-based strategy (EOSCS) can be summarized in the pseudo-code shown in Algorithm 2.

Algorithm 2. Cuckoo search with elite opposition-based strategy algorithm (EOSCS).

Begin
Objective function $f(x)$, $x = (x_1, x_2, \ldots, x_d)^T$
Generate initial population of n host nests $x_i = (i = 1, 2, \ldots, n)$ P_0
Generate opposition-based population of n host nests $oppo\_x_i = (i = 1, 2, \ldots, n)$ OP_0
Select the NP fittest individuals from $\{P_0 \cup OP_0\}$
Select the best individual as the elite individual X_e from $\{P_0 \cup OP_0\}$
While ($t <$ Max Generation) or (stop criterion)
Get a cuckoo $nest_i (i = 1, 2, \ldots, n)$ randomly by Lévy flights
Evaluate its quality/fitness F_i
Generate opposition-based nests $oppo\_nest_i (i = 1, 2, \ldots, n)$
For $i = 1{:}n$
$oppo\_nest_i = k \cdot (da_j + db_j) - X_e$;
End
Evaluate its quality/fitness $OPPO\_F_i$
If ($F_i > OPPO\_F_i$),
replace $nest_i$ by the new solution $oppo\_nest_i$;
replace F_i by the new quality/fitness $OPPO\_F_i$;
End
Choose a nest among n (say, j) randomly
If ($F_i > F_j$),
replace j by the new solution;
End
Dynamic Proportion Strategy **(DPS)** adjust p_a via formula (6);
A fraction (p_a) of worse nests are abandoned and new ones are built via **Local Neighborhood Search Strategy (LNSS)**;
Keep the best solutions (or nests with quality solutions);
Rank the solutions and find the current best
End while
Post-process results and visualization
End

4 Simulation Experiments

In this section, we applied 14 standard test functions to evaluate the optimal performance of CS with elite opposition-based strategy (EOSCS). The best, mean, worst, and Std. represent the optimal fitness value, mean fitness value, worst fitness value, and standard deviation (Std.), respectively. The result runs for each algorithm are summarized in Tables 1 and 2. The 14 standard benchmark functions have been widely used in the literature [11]. The dimensions, scopes, optimal values, and the iterations of 14 functions are in Table 3. We also do some high-dimensional tests, and the results are shown in Table 4. In addition, in order to compare the performance of the proposed approach and other improved CS algorithms, we used 28 more complex benchmark test functions from IEEE CEC 2013 competition [21] to test the 28 functions, and the results are summarized in Tables 5–8.

4.1 Experimental Setup

All the algorithms compared in this section are implemented in Matlab R2012a (7.14). Experiments are performed on a PC with a 3.01-GHz, AMD Athlon (tm) II X4 640 Processor, 3GB of RAM, and Windows XP operating system.

Table 1: Experiment Results of Benchmark Functions for Different Algorithms ($D = 30$).

Function	Result	CLSPSO	DE	BA	FOA	CS	EOSCS
F_1 ($D = 30$)	Best	9.00013E−16	6.915807234	0.000996122	2.13845E−05	0.004023394	2.3446E−249
	Mean	3.01674E−09	47.61180503	0.001354595	0.082713534	0.010066405	2.7017E−242
	Worst	2.80788E−08	177.693078	0.001693107	0.707014887	0.022198961	6.4823E−241
	Std.	5.66785E−09	51.60342989	0.000174248	0.216245998	0.00452777	0
	Rank	2	6	3	5	4	1
F_2 ($D = 30$)	Best	6.40057E−10	166.165864	15.16564042	0.00423111	64.02685333	0
	Mean	7.598673747	272.4051955	26.16697633	19.694717	86.43836671	0
	Worst	26	440.2251995	42.08389965	162.0176528	112.0529335	0
	Std.	6.077755021	62.16580992	6.511919575	43.74018031	12.91754674	0
	Rank	2	6	3	5	4	1
F_3 ($D = 30$)	Best	0	3998	0	0	0	0
	Mean	0	16139.63333	0.2	0	0.066666667	0
	Worst	0	66282	1	0	1	0
	Std.	0	17895.98025	0.406838102	0	0.253708132	0
	Rank	1	4	3	1	2	1
F_4 ($D = 30$)	Best	26.8326182	285.7097767	25.28054945	5.78746E−07	22.97272405	26.53121666
	Mean	28.41865876	3745.1064	27.77359091	32.43498523	25.35460187	27.59982225
	Worst	29.18401908	9363.006796	29.50148569	72.93680757	26.28696859	28.70536245
	Std.	0.481464337	3252.57872	1.193948427	13.82980311	0.644099778	0.495381107
	Rank	1	6	4	5	3	2
F_5 ($D = 30$)	Best	3.36602E−07	11.96744075	0.022867366	0.003407913	1.082011788	8.88178E−16
	Mean	2.70513E−05	16.42452995	0.999158775	0.250419879	1.983000642	8.88178E−16
	Worst	0.000115775	20.90579334	1.902688064	1.771753396	2.631160285	8.88178E−16
	Std.	2.85333E−05	2.610384328	0.652418731	0.572789366	0.349118062	0
	Rank	2	6	5	4	3	1
F_6 ($D = 30$)	Best	8.77076E−15	33.03291749	5.00523E−05	1.42564E−06	0.040447069	0
	Mean	1.38426E−10	153.7696637	7.20724E−05	0.010266246	0.107262004	0
	Worst	1.17837E−09	598.307314	8.57186E−05	0.140854303	0.178216749	0
	Std.	2.95817E−10	172.1359965	8.94179E−06	0.030247866	0.034612428	0
	Rank	2	6	3	4	5	1
F_7 ($D = 30$)	Best	0.099873346	5.728751666	0.299873346	0.099873346	1.025522743	0
	Mean	0.129873346	10.97147221	0.366540013	0.105513772	1.409106328	0
	Worst	0.199873346	25.66259745	0.499873346	0.237308406	1.77624218	0
	Std.	0.04660916	4.791025365	0.060647843	0.025017999	0.174585283	0
	Rank	3	6	4	2	5	1
F_8 ($D = 30$)	Best	1.19636E−05	0.680798801	0.015748487	0.001375633	0.015288963	2.93213E−06
	Mean	0.000296183	56.10768192	0.031328116	0.036668972	0.032166599	6.72564E−05
	Worst	0.00104468	162.3015255	0.054688111	0.811660473	0.048968339	0.000208014
	Std.	0.000207663	53.78178383	0.010703481	0.152286262	0.008024855	4.92982E−05
	Rank	2	6	4	5	3	1
F_9 ($D = 30$)	Best	6.03872E−09	13.93490571	0.013923243	0.002560367	4.710000811	3.4688E−126
	Mean	1.70153E−05	19.50407839	0.01659252	0.280985673	6.480231692	1.1993E−123
	Worst	6.70086E−05	31.8237867	0.020234424	1.789080575	7.97067096	8.7762E−123
	Std.	1.69009E−05	4.22469352	0.001523122	0.534253643	0.990702806	2.0236E−123
	Rank	2	6	3	4	5	1

4.2 Comparison of Each Algorithm Performance

The proposed EOSCS algorithm in this paper is compared with mainstream swarm intelligence algorithms CLSPSO, DE, BA, FOA, CS, respectively, using the best, mean, worst, and Stds. to compare their optimal performances. In this part, the population size is NP = 50. The experiment results are obtained in 30 trials. The setting values of algorithm control parameters of the mentioned algorithms are given below:

Table 2: Experiment Results of Benchmark Functions for Different Algorithms.

Function	Result	CLSPSO	DE	BA	FOA	CS	EOSCS
F_{10} ($D = 2$)	Best	−1.031628453	−1.031628453	−1.031628453	−0.866009808	−1.031628453	−1.031628453
	Mean	−1.031628453	−1.031628453	−1.031628416	−0.1488382	−1.031628453	−1.031628453
	Worst	−1.031628453	−1.031628453	−1.031628314	0.003090185	−1.031628453	−1.031628453
	Std.	6.51945E−16	6.77522E−16	3.49846E−08	0.265691693	6.77522E−16	6.77522E−16
	Rank	2	1	3	4	1	1
F_{11} ($D = 2$)	Best	27.70290555	0.397887358	0.397887358	0.84467413	0.397887358	0.397887358
	Mean	27.70290555	0.397887358	0.397887375	11.97252821	0.397887358	0.397887363
	Worst	27.70290555	0.397887358	0.397887417	51.99539967	0.397887358	0.397887448
	Std.	1.80672E−14	0	1.53224E−08	20.60147085	0	1.67207E−08
	Rank	2	1	3	5	1	4
F_{12} ($D = 2$)	Best	−123.5767709	−186.7309088	−186.7309073	−9.251536411	−186.7309088	−186.7309088
	Mean	−123.5767709	−186.4930224	−186.7308821	−1.52794067	−186.7309087	−186.7309088
	Worst	−123.5767709	−185.8356247	−186.73079	0.701000563	−186.7309083	−186.7309065
	Std.	1.86598E−14	0.230367245	2.568E−05	2.254032923	1.45146E−07	4.16234E−07
	Rank	1	5	4	6	2	3
F_{13} ($D = 2$)	Best	−3.03082E−05	−1	−1	−1	−1	−1
	Mean	−3.03082E−05	−1	−0.999999988	−0.7	−1	−1
	Worst	−3.03082E−05	−1	−0.999999957	−1.5E−08	−1	−1
	Std.	1.03382E−20	0	1.07244E−08	0.466091	0	0
	Rank	2	1	3	4	1	1
F_{14} ($D = 4$)	Best	−5.12847104	−10.53640982	−10.53618057	−10.5363	−10.53640982	−10.53640982
	Mean	−5.12847104	−10.5364	−5.445846498	−6.50465	−10.53640982	−10.53640982
	Worst	−5.12847104	−10.53640982	−3.835409293	−0.40396	−10.53640982	−10.53640982
	Std.	2.71009E−15	3.18E−15	1.4036311	5.022064	1.74269E−13	1.77636E−15
	Rank	2	3	5	6	4	1

1. CLSPSO parameters setting: weight factors $C_1 = C_2 = 1.49445$, $w\min = 0.4$, $w\max = 0.9$, $u = 3$, the population size is 50.
2. DE parameters setting: $F = 0.9$ and $CR = 0.9$, the population size is 50.
3. BA parameters setting: A = 0.25, $r = 0.5$, the population size is 50.
4. FOA parameters setting: the population size is 50.
5. CS parameters setting: the population size is 50, $Pa = 0.25$.
6. EOSCS parameters setting: the population size is 50, $Pa = 0.25$.

For the low-dimensional case, the maximum number of iterations of each algorithm is iterMax = 1000. From the rank of each function in Table 1, we can conclude that EOSCS provides the most of the better results than CS and other algorithms in low-dimensional case. What is more, EOSCS can find the theoretical optimum solutions for most benchmark functions (F_1, F_2, F_2, F_6, F_7, F_9) and has a very strong robustness. The number of finding the optimal solution for EOSCS is more than the other five algorithms. For other algorithms, CLSPSO and FOA can find only one theoretical optimum solution (F_3). DE is the worst algorithm compared with the other algorithms in low dimension in this paper. BA and CS do not have a strong robustness; the Std. is large.

From the rank of each function in Table 2, we can see that EOSCS provides better results than the other algorithms except F_{11}. For the most functions, EOSCS can find the theoretical optimum solutions (F_{10}, F_{12}, F_{13}, F_{14}). CS can find the theoretical optimum for $F_{10} \sim F_{13}$ except F_{14}. CLSPSO can find one theoretical optimum for F_{10}. As we can see that DE is better than BA in the low-dimensional case, it can find the theoretical optimum for $F_{10} \sim F_{14}$, BA can find the theoretical optimum values for $F_{10} \sim F_{13}$. FOA is the worst compared with the other algorithms in low dimension in this paper; it has low robustness and cannot find the theoretical optimum values.

For benchmark functions in Table 3, Figures 2–7 are the convergence curves. Figures 8–14 are the ANOVA tests of the global minimum. Figures 2–7 show that the convergence speed of EOSCS is quicker than the other

Table 3: Benchmark Test Functions.

Functions	Name	Benchmark functions	Scope	f_{min}
F_1	Sphere	$f(x)=\sum_{i=1}^{n} x_i^2$	[−100, 100]	0
F_2	Rastrigin	$f(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	[−5.12, 5.12]	0
F_3	Step	$f(x)=\sum_{i=1}^{n-1}(\lfloor x_i+0.5\rfloor)^2$	[−100, 100]	0
F_4	Rosenbrock	$f(x)=\sum_{i=1}^{n-1}[(x_i-1)^2+100(x_i^2-x_{i+1})^2]$	[−2, 2]	0
F_5	Ackley	$f(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right)\right)+20+e$	[−32, 32]	0
F_6	Griewank	$f(x)=\frac{1}{4000}\sum_{i=1}^{n}(x_i^2)-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[−600, 600]	0
F_7	Salomon	$f(x)=-\cos\left(2\pi\sqrt{\sum_{i=1}^{D}x_i^2}\right)+0.1\sqrt{\sum_{i=1}^{D}x_i^2}+1$	[−100, 100]	0
F_8	Quartic function, i.e. noise	$f(x)=\sum_{i=1}^{n}x_i^4+\text{random}(0, 1)$	[−1.28, 1.28]	0
F_9	Alpine	$f(x)=\sum_{i=1}^{n}\lvert x_i\sin(x_i)+0.1x_i\rvert$	[−10, 10]	0
F_{10}	Six hump camel back	$f(x_1, x_2)=\left(4-2.1x_1^2+\frac{x_1^4}{3}\right)x_1^2+x_1x_2+(-4+4x_2^2)x_2^2$	[−5, 5]	−1.03163
F_{11}	Branin	$f(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10$	[−5, 15]	0.397887
F_{12}	Shubert	$f(x)=\sum i\cos((i+1)x_1+i)\cdot\sum_{i=1}^{5}i\cos((i+1)x_2+i)$	[−10, 10]	−186.731
F_{13}	Easom	$f(x_1, x_2)=-\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2)$	[−100, 100]	−1
F_{14}	Shekel	$f(x)=-\sum_{i=1}^{m}\left(\sum_{j=1}^{n}[(x_j-a_{ij})^2+c_j]\right)^{-1}$, $m=30$	[0, 10]	−10.5364

algorithms. EOSCS is the best algorithm for most functions. Moreover, from the ANOVA tests of the global minimum, Figures 8–14 show that EOSCS is the most robust method. For the 14 benchmark functions, the other algorithms compared (i.e. CLSPSO, DE, BA, FOA, and CS) can only be robust for a few functions, but cannot be robust for all functions. Therefore, EOSCS is an effective and feasible method for optimization problems in the low-dimensional case.

In order to analyze the compared results, a nonparametric statistical significance proof known as the Wilcoxon's rank sum test [9] for independent samples has been conducted over the "average best-so-far" fitness value from Tables 1 and 2, with a 5% significance level. Tables 9 and 10 report the p-values produced by Wilcoxon's test for the pair-wise comparison of the mean fitness value of five groups. Such groups are formed by EOSCS versus CLSPSO, EOSCS versus DE, EOSCS versus BA, EOSCS versus FOA, and EOSCS versus CS. As a null hypothesis, it is assumed that there is no significant difference between the mean values of the two algorithms. The alternative hypothesis considers a significant difference between the mean fitness values of both approaches. From Table 9, we can see, except F_3, that the p-values reported in Table 9 are <0.05

Table 4: Experiment Results of Benchmark Functions for Different Algorithms (high Dimensional).

Function	Result	CLSPSO	DE	BA	FOA	CS	EOSCS
F_1 ($D = 1000$)	Best	2.00836E−11	7895.125118	3.761577797	0.000713885	65925.83	7.54693E−250
	Mean	0.175960628	8211.174157	4.034170666	3.80596881	79338.86	1.00000E−240
	Worst	5.278791352	8368.281929	4.318113678	23.90351134	92211.22	1.76470E−239
	Std.	0.96377086	117.3203901	0.142167645	8.66245771	6922.05	0
	Rank	3	5	2	4	6	1
F_2 ($D = 100$)	Best	2.42153E−08	1527.156141	45.85858786	0.01421204	401.8739	0
	Mean	35.06356514	1620.308015	100.9119234	46.34973847	458.6452	0
	Worst	97.99495906	1683.110692	139.0827724	384.5452066	509.4855	0
	Std.	33.87135946	41.07255346	21.59241331	120.4224467	30.68188	0
	Rank	4	5	2	6	3	1
F_3 ($D = 300$)	Best	0	832401	9	0	9031	0
	Mean	0.6	879182.7	23.83333333	0	11513.67	0
	Worst	3	915141	45	0	15273	0
	Std.	0.813676204	22738.21259	8.678484427	0	1309.508	0
	Rank	2	5	3	1	4	1
F_4 ($D = 1000$)	Best	988.9062883	406150.8255	1381.417884	3.39261E−05	4070.596	998.7547
	Mean	1459.277226	441080.8059	1495.434151	1008.31016	4494.842	998.86554
	Worst	3111.199433	456060.2859	1635.841829	2559.909396	5112.124	998.9545
	Std.	693.6565751	9825.681436	56.52071394	344.1476747	240.6719	0.051279
	Rank	5	6	2	4	3	1
F_5 ($D = 500$)	Best	2.22391E−07	21.09225837	0.625508921	0.003421386	9.979671	8.88178E−16
	Mean	2.36418E−05	21.16217443	1.42249258	0.097414102	10.39947	8.88178E−16
	Worst	0.000144058	21.1943914	2.042394157	1.512742533	10.78827	8.88178E−16
	Std.	2.9514E−05	0.022578636	0.279101315	0.358560524	0.229381	0
	Rank	2	3	5	6	4	1
F_6 ($D = 300$)	Best	7.88258E−15	7642.801047	0.002503168	2.29784E−06	65.83531	0
	Mean	3.09731E−10	7954.000363	0.003057476	0.008565204	85.99471	0
	Worst	2.23708E−09	8320.580971	0.003772345	0.096283085	106.2218	0
	Std.	5.30979E−10	169.1088133	0.000290272	0.026403975	9.99569	0
	Rank	2	6	3	4	5	1
F_7 ($D = 100$)	Best	2.71987E−06	50.71455341	0.499873346	0.099873346	6.399878	0
	Mean	0.129900509	52.75527946	0.663206679	0.161382292	7.201499	0
	Worst	0.299873346	55.74677	0.999873346	0.222482856	8.317335	0
	Std.	0.098748761	1.076140389	0.12172137	0.051257247	0.512042	0
	Rank	3	6	4	2	5	1
F_8 ($D = 500$)	Best	2.61005E−05	51607.41959	4.952381121	0.008711266	26.61293	9.10604E−06
	Mean	9.941846428	57496.08762	6.113680647	16.78431906	41.26846	6.21691E−05
	Worst	164.6189966	62575.14966	7.114303976	503.1941479	52.87461	0.000183
	Std.	32.23735119	2448.783416	0.543238482	91.86815017	6.73488	5.00780E−05
	Rank	4	6	2	5	3	1
F_9 ($D = 1000$)	Best	1.54254E−05	2778.080129	3.933394576	0.085035055	537.358	1.24949E−125
	Mean	0.001361727	2839.518289	4.209172491	3.673306348	604.4029	3.70078E−122
	Worst	0.013422419	2882.815973	5.16564886	38.60518789	648.516	3.35288E−121
	Std.	0.002468904	26.41486974	0.215590799	10.96421717	26.71147	7.29242E−122
	Rank	2	5	3	4	6	1

(5% significance level), which is strong evidence against the null hypothesis. It demonstrates that EOSCS performed better than CLSPSO, DE, BA, FOA, and CS considering eight functions (F_1, F_2, F_4, F_5, F_6, F_7, F_8, F_9), whereas from a statistical viewpoint, there is no difference between the results from EOSCS, CLSPSO, BA, FOA, and CS for F_3. In addition, the p-values of functions F_{10}, F_{11}, F_{12}, F_{13}, and F_{14} are shown in Table 10. It demonstrates that EOSCS performed better than CLSPSO, DE, BA, FOA, and CS considering the three functions F_{11}, F_{12} and F_{14}; however, from a statistical viewpoint, there is no difference between the results from CLSPSO, DE, and CS for F_{10} and F_{13}.

Table 5: CEC'2013 Test Functions.

Type	Functions	Function names	Optimal value
Unimodal functions	F_{15}	Sphere function	−1400
	F_{16}	Rotated high-conditioned elliptic function	−1300
	F_{17}	Rotated bent cigar function	−1200
	F_{18}	Rotated discus function	−1100
	F_{19}	Different powers function	−1000
Basic multimodal functions	F_{20}	Rotated Rosenbrock's function	−900
	F_{21}	Rotated Schaffers F7 function	−800
	F_{22}	Rotated Ackley's function	−700
	F_{23}	Rotated Weierstrass function	−600
	F_{24}	Rotated Griewank's function	−500
	F_{25}	Rastrigin's function	−400
	F_{26}	Rotated Rastrigin's function	−300
	F_{27}	Non-continuous rotated Rastrigin's function	−200
	F_{28}	Schwefel's function	−100
	F_{29}	Rotated Schwefel's function	100
	F_{30}	Rotated Katsuura function	200
	F_{31}	Lunacek Bi_Rastrigin function	300
	F_{32}	Rotated Lunacek Bi_Rastrigin function	400
	F_{33}	Expanded Griewank's plus Rosenbrock's function	500
	F_{34}	Expanded Scaffer's F6 function	600
Composition functions	F_{35}	Composition function 1 ($n = 5$, rotated)	700
	F_{36}	Composition function 2 ($n = 3$, unrotated)	800
	F_{37}	Composition function 3 ($n = 3$, rotated)	900
	F_{38}	Composition function 4 ($n = 3$, rotated)	1000
	F_{39}	Composition function 5 ($n = 3$, rotated)	1100
	F_{40}	Composition function 6 ($n = 5$, rotated)	1200
	F_{41}	Composition function 7 ($n = 5$, rotated)	1300
	F_{42}	Composition function 8 ($n = 5$, rotated)	1400
Search range:[−100, 100]			

4.3 High-Dimensional Function Test for 14 Benchmark Functions

In order to validate the optimization ability of algorithms in high-dimensional case, this paper set F_1, F_4, and F_9 to 1000-dimension, F_2 and F_7 to 100-dimension, F_3 and F_6 to 300-dimension, F_5 and F_8 to 500-dimension. The maximum number of iterations of each algorithm is adjusted to $iterMax = 1000$, and other parameters are the same.

The comparison results for the high-dimensional case are shown in Table 4. As seen from the results, the optimization performance of EOSCS is the best. Although the dimension of the functions is very high, the proposed algorithm (EOSCS) can also find optimum solutions for five benchmark functions (F_1, F_2, F_3, F_6, and F_7). Some algorithms can show a good performance in the low-dimensional case, but in the high-dimensional case, it cannot get a good result for most benchmark functions. The optimization ability of EOSCS does not show a significant decline with the increasing dimension.

Figures 15–23 are the convergence curves of the high-dimensional case, and Figures 24–32 are the ANOVA tests of the global minimum for the high-dimensional case. From the convergence curves in Figures 15–23, we can see that EOSCS is also the best algorithm for the most of the benchmark functions except F_3 and F_4. For F_3, the convergence speed of EOSCS is quick, and the convergence values are lower than CS, BA, and DE. For F_4, EOSCS is better than the other five algorithms except FOA. From the ANOVA tests of the global minimum, we can find that EOSCS is still the most robust method. Therefore, in the high-dimensional case, EOSCS can also be an effective and feasible method for optimization problems.

Table 6: Experiment Results of CEC'2013 Benchmark Functions ($D = 30$).

Functions	Result	CS	DECS	GCS	DDICS	CCS	EOSCS
F_{15} ($D = 30$)	Best	−1400	−1400	−1400	−1400	−1400	−1400
	Mean	−1400	−1400	−1400	−1400	−1400	−1400
	Worst	−1400	−1400	−1400	−1400	−1400	−1400
	Std.	0	0	0	1.2390603428132E−13	0	0
F_{16} ($D = 30$)	Best	2381862.33201953	73902.5654715794	2345017.17476891	936892.551072304	2932311.68489425	2878661.91153928
	Mean	5193551.55999065	428737.35590095	4647873.25778648	3370608.8401718	4672192.9019675	5098237.02574862
	Worst	9052258.05722615	1409012.33896018	7813391.27214963	5681756.67363522	8327835.84424977	7643663.37755482
	Std.	1625747.02975789	352486.191069062	1557714.99280245	636122.950894126	1268208.96069141	1233183.98746066
F_{17} ($D = 30$)	Best	1755676.41563328	−919.305644458268	773.961434606	23604228.4713579	2475510.20713489	16815562.5806191
	Mean	25113149.6648338	13120033.1739245	5524627.61761353	132716108.186056	14466214.8327562	72525692.6640158
	Worst	101509976.968131	141068880.17503	30754020.9862811	302158302.291013	61906344.8946426	188189971.402934
	Std.	22636705.6492686	30974783.4907728	7268427.35881605	90254281.3010472	14274045.058204	49609549.4598715
F_{18} ($D = 30$)	Best	24250.2529420408	−1099.54404326646	14829.2775977932	49226.8610786085	19165.1209715351	14584.1884857572
	Mean	40777.395128242	−1061.90995651896	22767.7587699555	74058.2463041902	27393.5759345419	24178.400109653
	Worst	55324.0508640976	−946.516184199849	32690.9259413721	87021.5508645308	36196.1332992866	34049.504748933
	Std.	7646.83426480011	37.9682109222809	4666.20061275346	10392.1083346908	3996.30687972905	3528.10989266443
F_{19} ($D = 30$)	Best	−1000	−1000	−1000	−1000	−1000	−1000
	Mean	−1000	−1000	−1000	−1000	−1000	−1000
	Worst	−1000	−1000	−999.999999999995	−1000	−1000	−1000
	Std.	1.09696575390673E−13	0	9.02114314415051E−13	5.06752089636162E−13	8.44444631079804E−13	1.0803982029575E−13

Table 7: Experiment Results of CEC'2013 Benchmark Functions ($D = 30$).

Functions	Result	CS	DECS	GCS	DDICS	CCS	EOSCS
F_{20} ($D = 30$)	Best	−895.929246180825	−888.580733393116	−898.019059241603	−896.339056017631	−895.780437452659	−894.201509601429
	Mean	−885.018348647075	−885.58665193975	−886.465000187078	−888.704220445099	−886.336681168628	−889.70159088683
	Worst	−877.715350469658	−882.91659847323	−880.429775250757	−881.970704345841	−880.014014775079	-878.311872780083
	Std.	5.2923547149061	1.29877038846978	3.6537874530819	6.2210680434918	4.09214087673617	3.98621201242752
F_{21} ($D = 30$)	Best	−721.263145031941	−788.430268119815	−762.665309869151	−732.268650048002	−753.558937060604	−713.151700141913
	Mean	−679.049508030357	−687.199943786849	−750.989148485299	−692.207882295147	−727.867474110142	−684.052040639882
	Worst	−640.249508030357	−487.672152484315	−719.049555912	−661.107961810299	−695.732361091465	−646.841220175723
	Std.	18.2644543489303	85.5534143360776	10.2905118228678	16.0266320150084	15.2183424820465	17.0140745262864
F_{22} ($D = 30$)	Best	−679.156760790011	−679.229853642106	−679.225535782403	−679.006374863128	−679.210408004552	−679.223263872758
	Mean	−679.076758926417	−679.079218250199	−679.085729824302	−679.068602342812	−679.079469617406	−679.106567878927
	Worst	−679.011220965389	−678.98060426087	−678.998944050979	−679.108272646023	−679.008107568138	−679.008594406186
	Std.	0.0369546965775582	0.0481326282546476	0.0483822915039498	0.0380621136708024	0.0547877559836104	0.0368887748607162
F_{23} ($D = 30$)	Best	−573.74731947972	−587.890280198903	−579.012974181265	−577.38032910266	−576.860223232522	−578.569461138026
	Mean	−569.966843569511	−568.680795984224	−574.636946137911	−573.562097202209	−574.492849276935	−575.045845324053
	Worst	−566.931991789796	−557.64865001806	−571.884782020532	−565.140533912582	−571.685896675813	−570.140747796136
	Std.	1.50711109427152	8.43209650849181	1.45667605627812	1.57082562021089	1.28298051451451	1.09958244002623
F_{24} ($D = 30$)	Best	−499.999998804027	−500	−499.997539618028	−499.995502553209	−499.999995942681	−499.999999728869
	Mean	−499.998259005586	−499.974463742565	−499.990040793712	−499.940819912688	−499.995658133634	−499.997171462854
	Worst	−499.992434499648	−499.933435637739	−499.971598202767	−499.88109962684	−499.986168904711	−499.987449459271
	Std.	0.00300983663150767	0.0194329383672745	0.0054340683270514	0.0367635923692072	0.00418496899980589	0.00403810870278092
F_{25} ($D = 30$)	Best	−370.591305154735	−387.065537295606	−380.897159712754	−362.620516203913	−384.804102897759	−400
	Mean	−358.447919867389	−359.770540056346	−373.804076819589	−346.340215495806	−375.281140852324	−400
	Worst	−344.337453765538	−321.398360525983	−359.891162662914	−322.685862760128	−363.618610506613	−400
	Std.	5.87181888603657	12.8369026791841	4.85266115016928	10.8512303980722	4.97793909941583	0
F_{26} ($D = 30$)	Best	−200.97659240664	−269.156289381384	−259.0373349408	−80.2484932059668	−234.471400142381	−200.538917070746
	Mean	−141.548035953821	−220.960944731843	−241.900141893094	−13.82608495002158	−210.033522685549	−150.714698326961
	Worst	−102.880081643581	−180.846042607707	−219.299288642335	38.2550962619264	−188.907732514307	−110.992754269373
	Std.	24.7102461027066	26.0335997477962	9.01396212836221	31.6803846673589	11.3245819034618	25.9503771691543
F_{27} ($D = 30$)	Best	−72.0203993162767	−147.497400196765	−121.245155788373	92.6828810246288	−110.238378181538	−37.955355654476
	Mean	−21.2130646445268	−88.4479874685669	−91.0876269321563	135.830122988623	−57.0656583402014	−13.3351968725798
	Worst	19.0583998183353	−37.166954529235	−65.6898197725726	188.034690244834	−28.7315986423396	36.7073389954337
	Std.	26.1164716496537	26.0536861016145	12.8300199364542	28.6027446503562	16.7613595595025	18.142064159204
F_{28} ($D = 30$)	Best	1573.47761473746	1141.19343100582	2414.18583205804	1492.13027981229	1538.70284306737	−100
	Mean	2133.44029868326	2769.04965660613	2963.88094691816	2268.320128968	2081.05724185645	−100
	Worst	2544.54095436292	4453.19568245005	3404.14330781414	2910.23221854398	2508.86695934366	−100
	Std.	239.547619172801	696.480209111839	266.137345698842	302.590634302258	204.0894541365	0

Table 7 (continued)

Functions	Result	CS	DECS	GCS	DDICS	CCS	EOSCS
F_{29} ($D = 30$)	Best	3779.64417961009	2040.8117097699	3243.66814827731	3128.06823231557	3578.73336146489	3450.19973013774
	Mean	4449.72856657124	5181.39641599787	3880.36288257749	3682.03299088532	4072.37573102033	3637.23901979714
	Worst	4934.56816501619	7119.087472271	4757.63462254209	4186.24352342843	4444.3965296603	4018.34400727106
	Std.	265.432847199367	1481.47027130861	335.979182738031	325.053020547908	235.838206911141	207.721984233849
F_{30} ($D = 30$)	Best	201.420934680344	200.862116495325	201.289382500128	200.50936235269	201.378094498127	200
	Mean	201.888506670315	202.282692405723	201.38454581259	200.662406777224	201.800936646516	200
	Worst	202.386384174437	202.785573551989	202.433255908002	200.855469994658	202.267702394484	200
	Std.	0.230002249172378	0.431065964739641	0.272707761717345	0.152914083802324	0.248711782665156	0
F_{31} ($D = 30$)	Best	393.730699957154	347.517374843732	367.960434228024	330.433748333473	369.656507817633	423.187509325952
	Mean	425.01012577321	372.432299400272	382.641120780895	330.433748333473	380.863983536402	454.233951799564
	Worst	457.160737364568	406.710333218157	399.072475363552	330.433748333474	399.341092327124	486.302225858418
	Std.	17.068380096378	11.7228219640929	7.60493136204118	1.0613812849682E−13	7.08346412713968	13.6820855632363
F_{32} ($D = 30$)	Best	545.391814182264	455.638828418253	477.239395445063	592.326805521489	496.719592821735	565.122610115661
	Mean	578.654105569593	507.79061617333	498.265288740508	641.024252731328	517.175545170255	596.921554490808
	Worst	625.653300989547	586.194262004413	518.749181185008	718.461489154862	534.253035412314	632.634158451172
	Std.	20.8330863016056	46.1119606729556	9.56159544400562	30.6972312719088	9.18978047570176	17.5165250435902
F_{33} ($D = 30$)	Best	504.699581460036	502.101313704461	503.735915672875	506.326656831508	507.384734463687	503.480971213154
	Mean	507.414722327516	504.275818258747	505.093184992292	509.820004798726	510.118787325047	504.621755273792
	Worst	510.133344826769	509.519317739317	506.368557851216	513.38218638908	513.174173842688	505.626954313866
	Std.	1.19285611779306	1.4895814746536	0.603548977749196	1.44641328923992	1.55145744153346	0.582611869975249
F_{34} ($D = 30$)	Best	612.275041272816	612	611.517822211132	612.903370247072	611.453217092515	610.877277223184
	Mean	612.852389999102	614.71893371228	612.75267023189	614.329015696282	612.169720576226	611.633738299532
	Worst	613.470739067652	615	613.252375268846	615.499950563378	612.568915340116	612.329715066012
	Std.	0.315503509582117	0.700633447197597	0.335524191931454	0.553605847141266	0.332488832061551	0.30800187314014

Table 8: Experiment Results of CEC'13 Benchmark Functions ($D = 30$).

Function	Result	CS	DECS	GCS	DDICS	CCS	EOSCS
F_{35} ($D = 30$)	Best	800.00024944965	900	900	800.682181602894	895.302473757405	800.826052566579
	Mean	890.090370281559	1040.75098999727	999.395544160888	838.194800588429	924.825554158604	881.458205690583
	Worst	900.040055827297	1143.54414165985	1143.54414165985	902.000000261402	1000	901.030472470572
	Std.	30.2450873596026	92.3710194300076	75.4971651025115	48.720475623655	35.0438636738336	38.1180183297286
F_{36} ($D = 30$)	Best	3075.80391468567	1928.98252969874	3773.38090584296	3266.31968596426	2601.02255313682	2816.03157206464
	Mean	3617.99727759839	3957.52915094843	4797.94381693328	3652.38300015322	3569.33576174617	3563.21758869271
	Worst	4035.37509611334	5362.25428399717	6059.56148622925	4088.62748031502	3907.93372563586	4284.94333790954
	Std.	227.324789716795	877.328138501594	540.811942695814	236.410709259829	292.345156606008	367.849934894909
F_{37} ($D = 30$)	Best	5396.36743160712	3961.41293098172	4514.58553068277	4827.15890526886	4980.30362758655	5017.39750678403
	Mean	6385.25234592529	7159.60072461038	5203.02137587693	5526.34682072822	5540.4940845052	5812.85496743739
	Worst	6934.48516933062	8255.80809844845	5887.34313103926	6019.40903829866	6141.36999133545	6834.2732301738
	Std.	341.366511316293	1165.94740377515	308.212999335496	431.387006159161	264.692498763443	464.814691330387
F_{38} ($D = 30$)	Best	1278.07656850974	1233.34643495506	1273.40635306316	1266.92021532438	1264.64266687099	1269.56659793886
	Mean	1286.48492342914	1285.49189193609	1281.4275939737	1276.98656075834	1282.71976285332	1276.2500074952
	Worst	1292.90508672726	1297.83410480124	1287.17043514468	1289.01427647008	1294.61745212667	1287.06987613321
	Std.	3.60795769297181	15.4684357513082	3.3191539689625	5.93090316375022	6.3325674205439	3.64333632763657
F_{39} ($D = 30$)	Best	1381.06882037601	1375.80561418128	1383.2388691026	1364.86277381423	1381.27619234681	1372.48788715007
	Mean	1392.01265043203	1394.36918314899	1391.70510651295	1378.55967744123	1390.68748131431	1377.36951820482
	Worst	1398.82366842832	1401.22743011512	1397.45406444499	1385.57226286408	1402.41977426623	1387.95705211961
	Std.	4.77480916264684	5.75416801949107	3.56722894644083	5.6231690834212	4.90142931459739	3.68438552518675
F_{40} ($D = 30$)	Best	1400.13505968564	1400.03448588834	1400.06196558506	1400.08522436928	1400.12383098643	1400.26663288324
	Mean	1401.04798962862	1521.98022069364	1400.15136450072	1436.84405703486	1400.26039488235	1435.44683212998
	Worst	1408.16908213731	1586.32960130074	1400.3207945082	1580.52372045496	1401.17265802383	1579.69330548572
	Std.	1.65167336453624	73.8389375562287	0.0590461218873901	54.8925332044604	0.187274393338571	70.3326094332151
F_{41} ($D = 30$)	Best	1985.91474769828	1774.10165234229	2220.86588449673	1746.10563330358	1812.79930395679	2251.46077382748
	Mean	2411.00338718567	2338.21507528694	2299.69211035936	1978.37726023753	2303.62293662969	2379.77759082512
	Worst	2502.1506007865	2566.90586716956	2404.67848515805	2453.22684904361	2432.45903334117	2452.14231063871
	Std.	90.501727232759	267.264308070254	51.6528127261409	340.09605277522	135.284493349371	50.2815618619341
F_{42} ($D = 30$)	Best	1700.00000000001	1700	1700	1700	1700	1700
	Mean	1700.00000001862	1700	1693.33333471062	1664.20868505332	1692.98205083833	1700
	Worst	1700.0000005567	1700	1500	1500.01523960184	1500	1700
	Std.	1.0162744750794E−07	2.31260686503268E−13	36.5148296232802	80.2317900012592	36.468300886251	1.33984114578127E−13

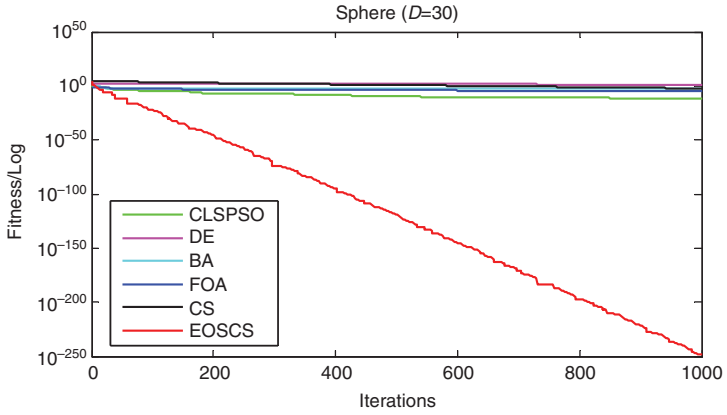

Figure 2: Convergence Curves for F_1 ($D = 30$).

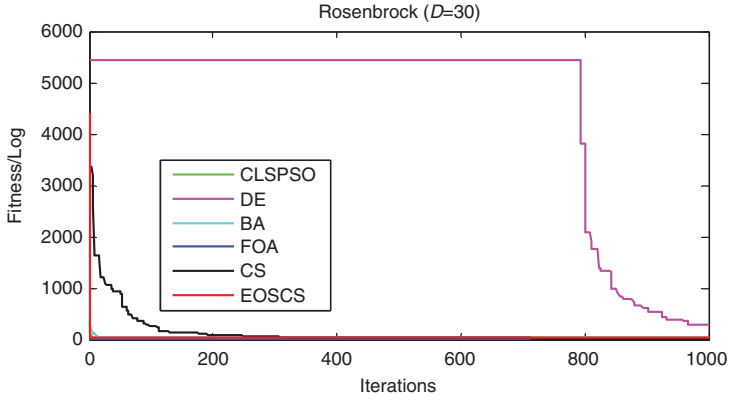

Figure 3: Convergence Curves for F_4 ($D = 30$).

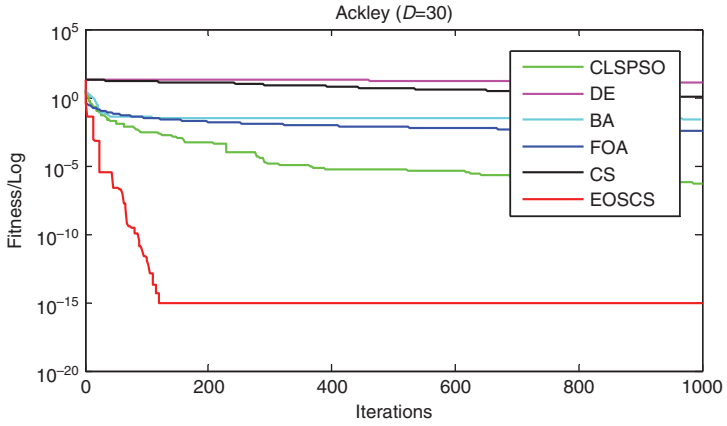

Figure 4: Convergence Curves for F_5 ($D = 30$).

The Wilcoxon rank test results of the high-dimensional case, which are presented in Table 11, demonstrate that EOSCS performed better than DE, BA, DE, and CS considering night functions $F_1 - F_9$. However, there is no difference between the results from EOSCS, CLSPSO, and FOA for F_3 and F_4.

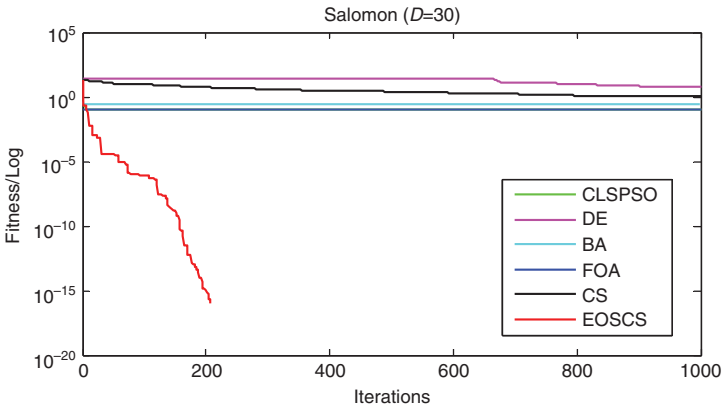

Figure 5: Convergence Curves for F_7 ($D = 30$).

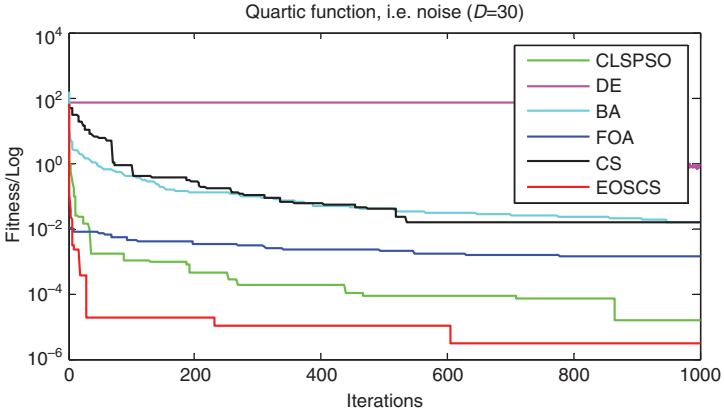

Figure 6: Convergence Curves for F_8 ($D = 30$).

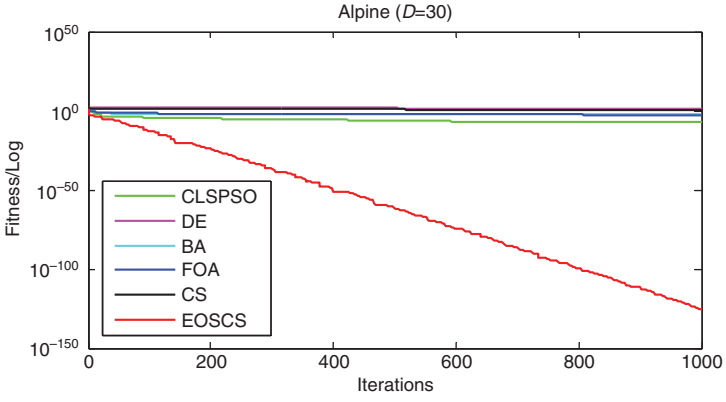

Figure 7: Convergence Curves for F_9 ($D = 30$).

4.4 Compared with Other Improved Cuckoo Search Algorithms

In order to validate the optimization ability of EOSCS, the proposed algorithm is compared with other recent CS algorithms (CS, CCS [35], DDICS [34], DECS [33], GCS [42]). In this part, we use 28 more complex benchmark functions from the IEEE CEC'2013 competition [21]. These functions are shown in Table 5. In this part,

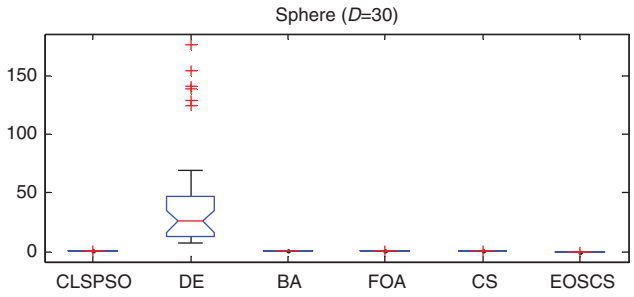

Figure 8: ANOVA Tests of The Global Minimum for F_1 ($D = 30$).

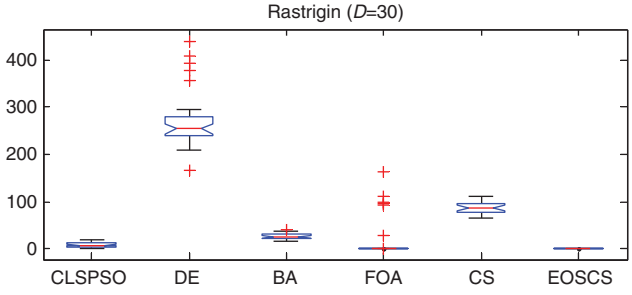

Figure 9: ANOVA Tests of The Global Minimum for F_2 ($D = 30$).

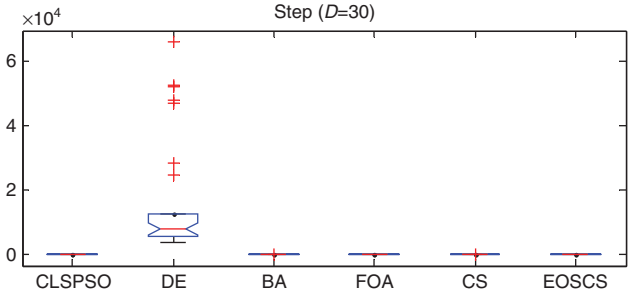

Figure 10: ANOVA Tests of The Global Minimum for F_3 ($D = 30$).

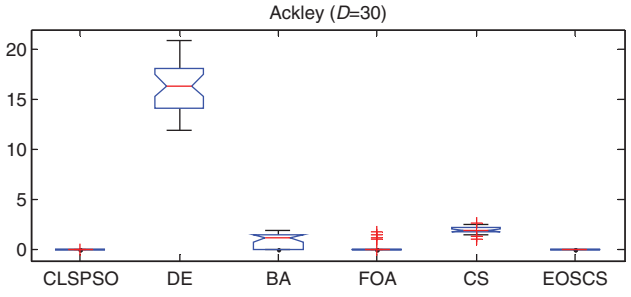

Figure 11: ANOVA Tests of The Global Minimum for F_5 ($D = 30$).

the results are obtained in 30 trials, the dimension is 30, the maximum number of iterations of each algorithm is adjusted to $iterMax = 5000$. The setting values of the algorithm control parameters of the mentioned algorithms are given below:

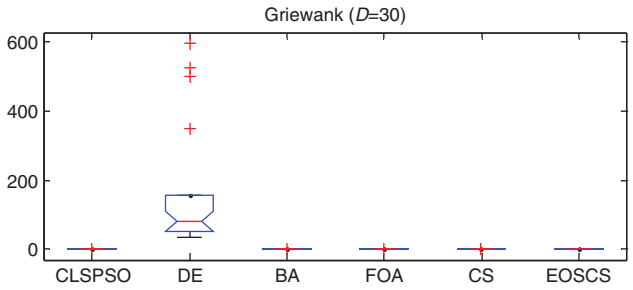

Figure 12: ANOVA Tests of The Global Minimum for F_6 ($D = 30$).

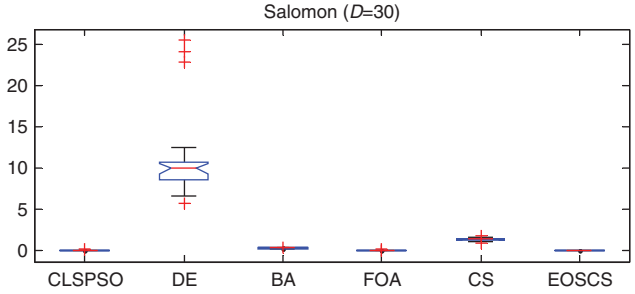

Figure 13: ANOVA Tests of The Global Minimum for F_7 ($D = 30$).

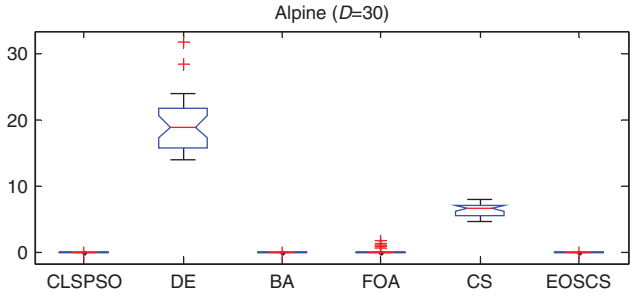

Figure 14: ANOVA Tests of The Global Minimum for F_9 ($D = 30$).

Table 9: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CLSPSO, EOSCS versus DE, EOSCS versus BA, EOSCS versus FOA, and EOSCS versus CS Over The "Average Best-So-Far" Values from Table 1.

					EOSCS VS
	CLSPSO	DE	BA	FOA	CS
F_1	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_2	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_3	1	1.7344e−006	0.0313	1	0.5000
F_4	2.8786e−006	1.7344e−006	0.1846	2.3704e−005	1.7344e−006
F_5	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_6	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_7	3.2578e−007	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_8	6.3391e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_9	1.7344e−006	1.7344e−006	1.7344e−006	1.6944e−006	1.7344e−006

Table 10: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CLSPSO, EOSCS versus DE, EOSCS versus BA, EOSCS versus FOA, and EOSCS versus CS Over The "Average Best-So-Far" Values From Table 2.

					EOSCS VS
	CLSPSO	DE	BA	FOA	CS
F_{10}	0.1250	1	1.7344e−006	1.7344e−006	1
F_{11}	1.7344e−006	1.7344e−006	2.6134e−004	1.7344e−006	1.7344e−006
F_{12}	1.7181e−006	1.9209e−006	1.7344e−006	1.7344e−006	1.7423e−004
F_{13}	4.3205e−008	1	1.7344e−006	1.7344e−006	1
F_{14}	6.7988e−008	9.6336e−007	1.7344e−006	1.7344e−006	1.7159e−006

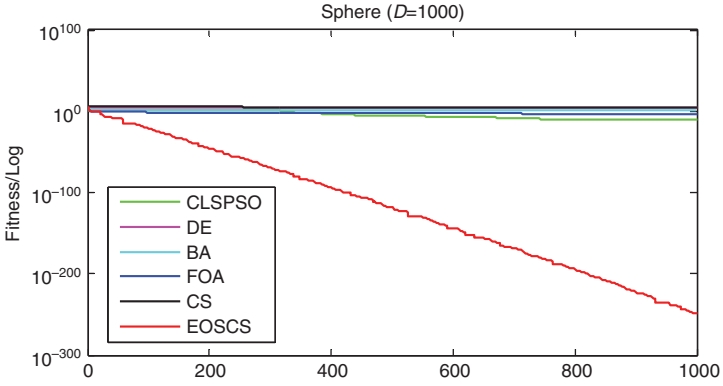

Figure 15: Convergence Curves for F_1 ($D = 1000$).

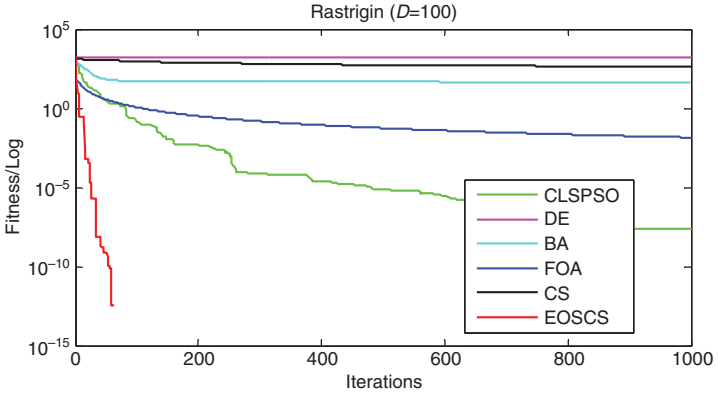

Figure 16: Convergence Curves for F_2 ($D = 100$).

In CCS, the probability of abandoned $Pa = 0.25$, the population size is 50, the chaotic factor is $\mu = 3$. In GCS, the probability of abandoned $Pa = 0.25$, the population size is 50. In DECS, the probability of abandoned $Pa = 0.25$, variation factor $F = 0.9$, the population size is 50. In DDICS, the probability of abandoned $Pa = 0.25$, the population size is 50.

The comparison results are shown in Tables 6–8, including the functions that are used in the CEC'2013 contest. Using such functions, the EOSCS algorithm is compared to CS, CCS, DDICS, DECS, and GCS. The comparison results of the unimodal functions, basic multimodal functions, and composition functions are shown in Tables 6–8, respectively. Likewise, the *p*-values of the Wilcoxon signed-rank test of 30 independent executions are listed in Tables 12–14.

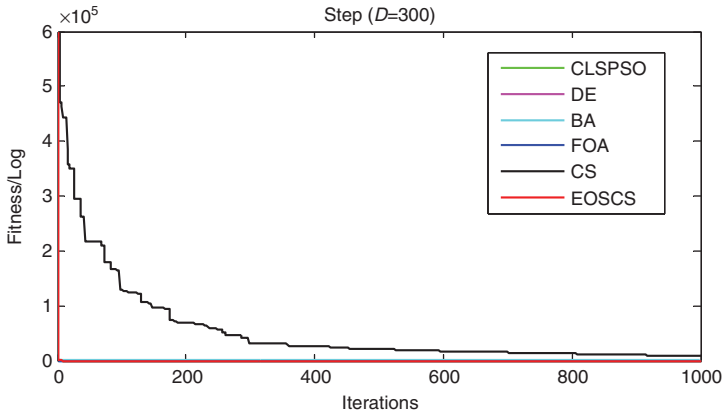

Figure 17: Convergence Curves for F_3 ($D = 300$).

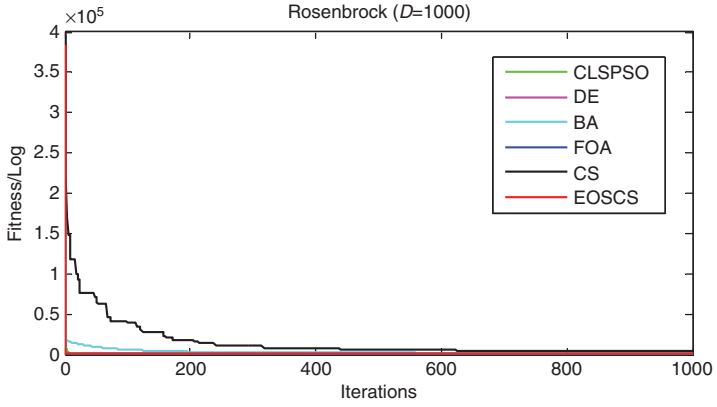

Figure 18: Convergence Curves for F_4 ($D = 1000$).

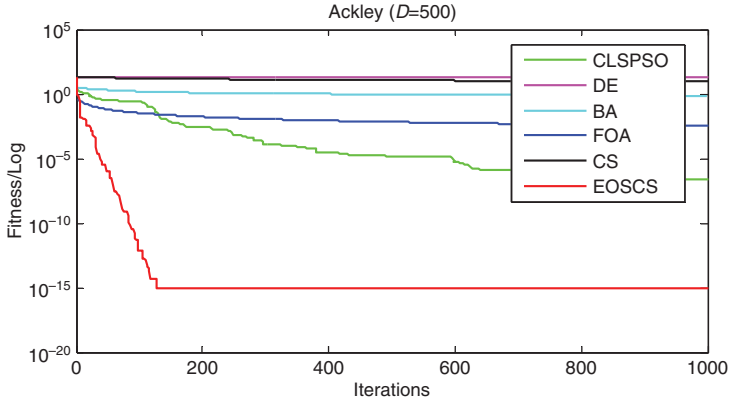

Figure 19: Convergence Curves for F_5 ($D = 500$).

According to the compared results of Table 6, for F_{15}, F_{18}, and F_{19}, the mean fitness value and the Std. of EOSCS are better than those of the other algorithms. See that from Table 7, for F_{20}, F_{22}, F_{23}, F_{24}, F_{29}, F_{30}, F_{33}, and F_{34}, the mean fitness value and the Std. of EOSCS are also better than those of the other algorithms. As shown in Table 8, for F_{35}, F_{36}, F_{38}, F_{39} and F_{42}, the mean fitness values are the best compared with those of the other

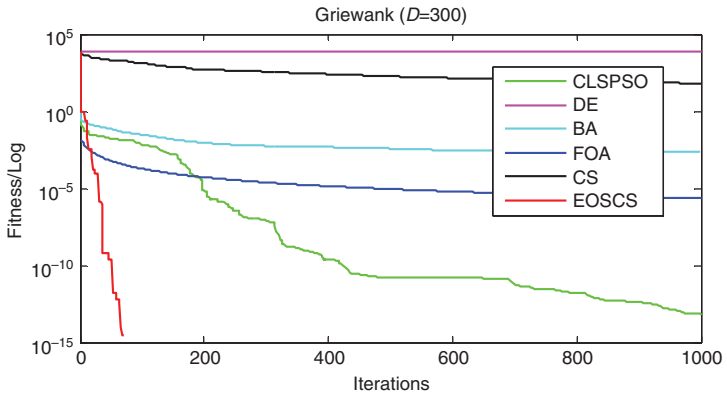

Figure 20: Convergence Curves for F_6 ($D = 300$).

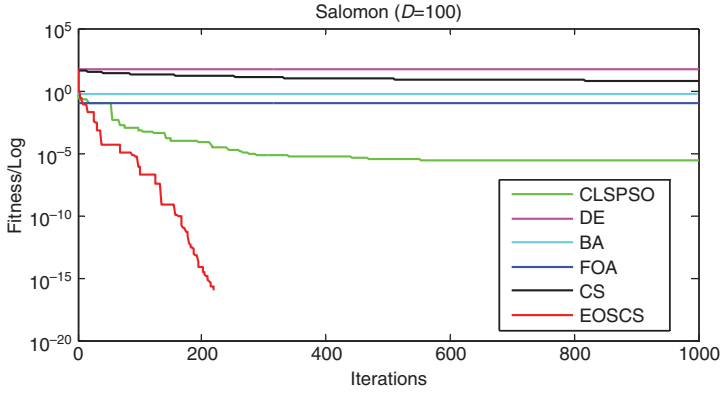

Figure 21: Convergence Curves for F_7 ($D = 100$).

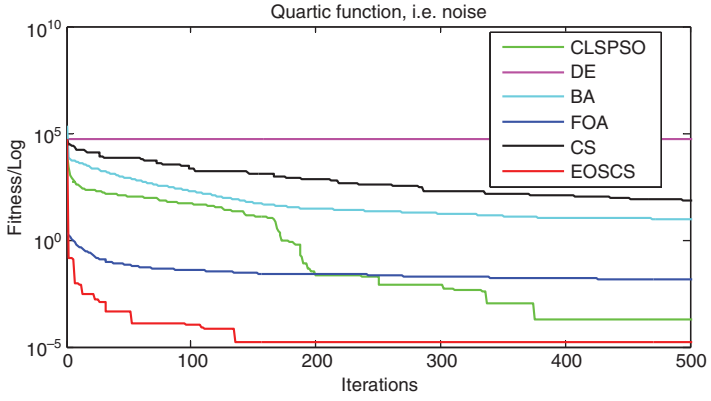

Figure 22: Convergence Curves for F_8 ($D = 500$).

algorithms (CS, CCS, GCS, DECS, and DDICS). It is evident that EOSCS yields much better solutions than the other methods in most of the functions.

The Wilcoxon rank test results of the unimodal functions ($F_{15} - F_{19}$), which are presented in Table 12, demonstrate that EOSCS performed better than CS, CCS, GCS, DECS, and DDICS considering the five functions $F_{15} - F_{19}$, whereas from a statistical viewpoint, there is no difference between the results from EOSCS, CS, CCS, and DECS for F_{15}; in addition, there is no difference between the results from EOSCS and CS for F_{16}. For the

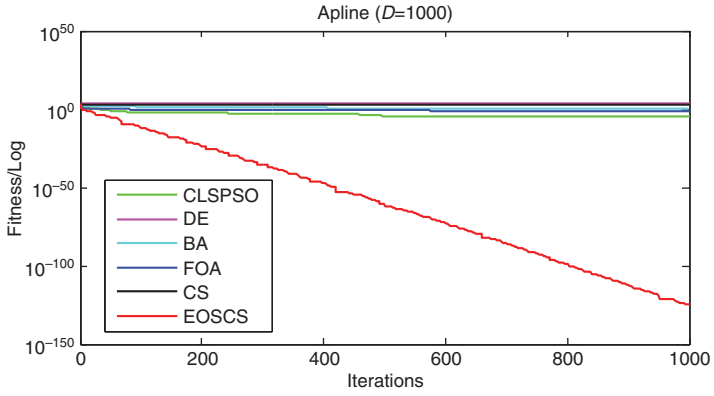

Figure 23: Convergence Curves for F_9 ($D = 1000$).

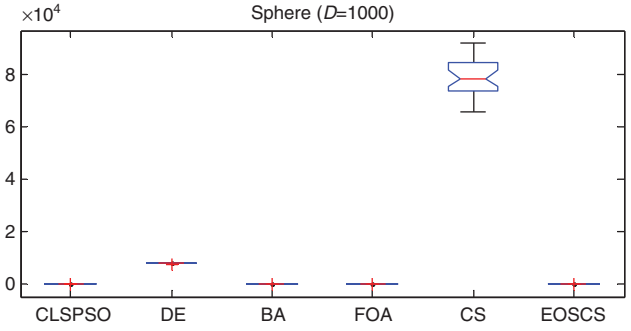

Figure 24: ANOVA Tests of The Global Minimum for F_1 ($D = 1000$).

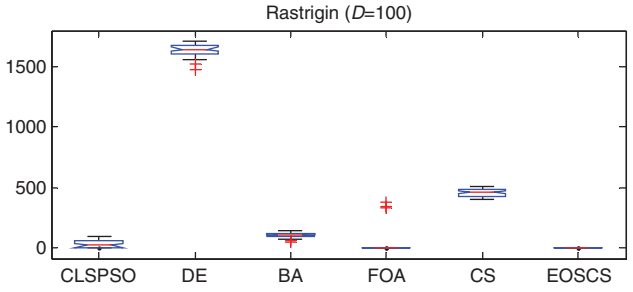

Figure 25: ANOVA Tests of The Global Minimum for F_2 ($D = 100$).

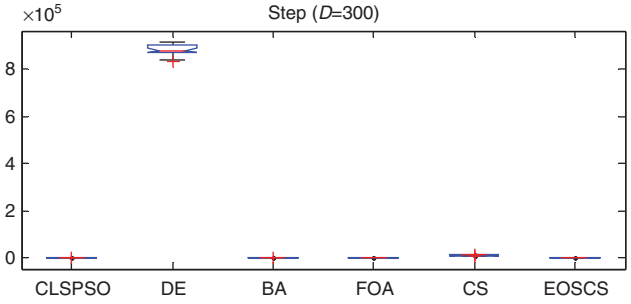

Figure 26: ANOVA Tests of The Global Minimum for F_3 ($D = 300$).

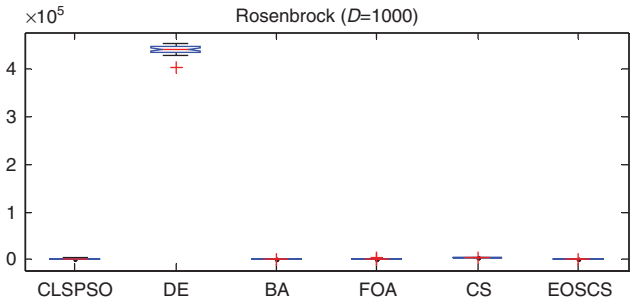

Figure 27: ANOVA Tests of The Global Minimum for F_4 ($D = 1000$).

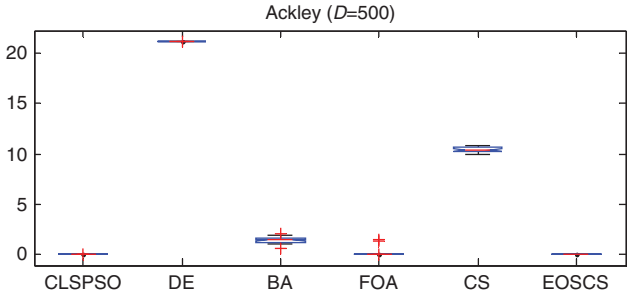

Figure 28: ANOVA Tests of The Global Minimum for F_5 ($D = 500$).

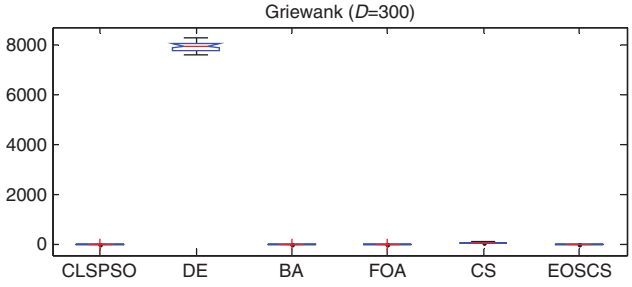

Figure 29: ANOVA Tests of The Global Minimum for F_6 ($D = 300$).

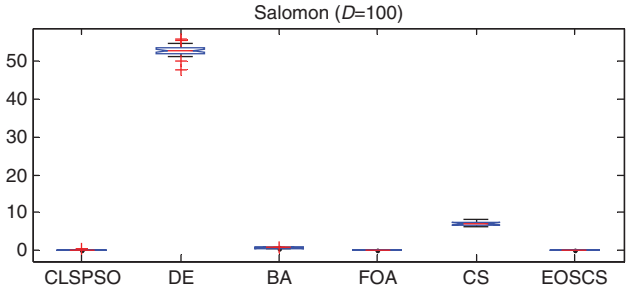

Figure 30: ANOVA Tests of The Global Minimum for F_7 ($D = 100$).

basic multimodal functions and composition functions ($F_{20} - F_{42}$), the Wilcoxon test results in Tables 13 and 14 provide information to statistically demonstrate that EOSCS has performed better than CS, CCS, DDICS, DECS, and GCS, mostly.

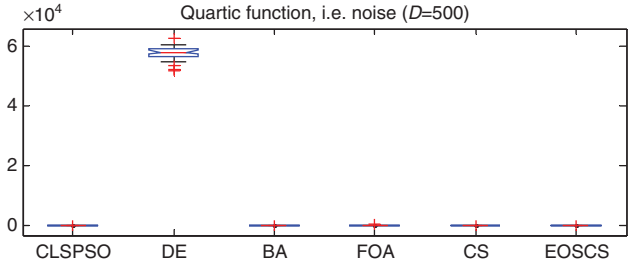

Figure 31: ANOVA Tests of The Global Minimum for F_8 ($D = 500$).

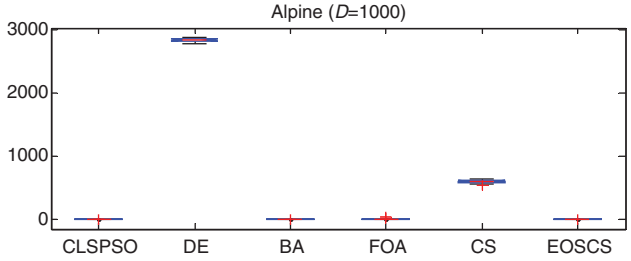

Figure 32: ANOVA Tests of The Global Minimum for F_9 ($D = 1000$).

Table 11: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CLSPSO, EOSCS versus DE, EOSCS versus BA, EOSCS versus FOA, and EOSCS versus CS Over The "Average Best-So-Far" Values from Table 4 (high dimensional).

	EOSCS VS				
	CLSPSO	**DE**	**BA**	**FOA**	**CS**
F_1 ($D = 1000$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_2 ($D = 100$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_3 ($D = 300$)	2.4414e−004	1.7344e−006	1.7235e−006	1	1.7344e−006
F_4 ($D = 1000$)	0.3820	1.7344e−006	1.7344e−006	3.1123e−005	1.7344e−006
F_5 ($D = 500$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_6 ($D = 300$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_7 ($D = 100$)	1.2598e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_8 ($D = 500$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006
F_9 ($D = 1000$)	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006	1.7344e−006

Table 12: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CS, EOSCS versus CCS, EOSCS versus GCS, EOSCS versus DECS, and EOSCS versus DDICS Over the "Average Best-So-Far" Values from Table 6.

	EOSCS VS				
	CS	**CCS**	**GCS**	**DECS**	**DDICS**
F_{15}	1	1	1	1	2.0346e−07
F_{16}	0.3493	0.0087	0.0407	1.7344e−06	1.7344e−06
F_{17}	3.7243e−05	6.3391e−06	1.7344e−06	3.1123e−05	0.0023
F_{18}	0.0157	0.0030	1.3601e−05	1.7344e−06	1.7344e−06
F_{19}	0.0156	0.0039	0.2344	4.8828e−04	1.2132e−07

Table 13: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CS, EOSCS versus CCS, EOSCS versus GCS, EOSCS versus DECS, and EOSCS versus DDICS Over the "Average Best-So-Far" Values from Table 7.

					EOSCS VS
	CS	CCS	GCS	DECS	DDICS
F_{20}	0.9426	0.1306	0.0166	0.1529	7.7122e–04
F_{21}	0.0316	1.7344e–06	1.7344e–06	0.4779	0.2134
F_{22}	0.0230	0.0140	0.0786	0.0978	1.4936e–05
F_{23}	0.0218	6.3391e–06	1.1265e–05	0.0368	0.5716
F_{24}	0.1306	0.1359	2.5967e–05	3.8822e–06	1.7344e–06
F_{25}	7.6909e–06	1.7344e–06	1.7344e–06	1.1265e–05	1.7344e–06
F_{26}	0.2989	1.7344e–06	1.7344e–06	1.7344e–06	1.7344e–06
F_{27}	0.2712	1.7344e–06	1.7344e–06	1.7344e–06	1.7344e–06
F_{28}	0.4048	0.0157	1.7344e–06	5.2872e–04	1.7344e–06
F_{29}	6.6392e–04	0.2623	0.0077	0.0053	4.0715e–05
F_{30}	0.1529	1.7344e–06	0.3820	8.9187e–05	1.7344e–06
F_{31}	9.3157e–06	1.7344e–06	1.7344e–06	1.7344e–06	1.7344e–06
F_{32}	0.0026	1.7344e–06	1.7344e–06	2.1266e–06	1.3601e–05
F_{33}	4.7292e–06	1.7344e–06	1.7344e–06	1.7344e–06	1.7344e–06
F_{34}	0.4284	5.7517e–06	1.9209e–06	2.3534e–06	1.7344e–06

Table 14: *p*-Values Produced by Wilcoxon's Test Comparing EOSCS versus CS, EOSCS versus CCS, EOSCS versus GCS, EOSCS versus DECS, and EOSCS versus DDICS Over The "Average Best-So-Far" Values from Table 8.

					EOSCS VS
	CS	CCS	GCS	DECS	DDICS
F_{35}	0.0897	6.1564e–04	7.6909e–06	7.6909e–06	9.7110e–05
F_{36}	0.5304	0.0519	2.6033e–06	0.0472	1.7344e–06
F_{37}	1.4936e–05	0.0098	2.5967e–05	5.7924e–05	0.0030
F_{38}	3.8822e–06	0.0082	0.0125	0.0034	0.7036
F_{39}	3.1817e–06	1.9209e–06	2.1266e–06	2.1266e–06	5.2165e–06
F_{40}	0.0028	1.7344e–06	1.7344e–06	1.0570e–04	1.7344e–06
F_{41}	0.0050	0.0012	6.3198e–05	0.9099	1.7988e–05
F_{42}	3.1123e–05	2.8434e–05	2.8434e–05	3.1123e–05	1.6046e–04

5 Conclusions

In this paper, the CS algorithm EOS, local neighborhood search strategy, and dynamic proportion strategy are proposed for enhancing its performance. The opposite solution of the elite individual in the population is generated by an opposition-based strategy in the proposed algorithm, and a generalized coefficient k is introduced to form an opposite search space by constructing the opposite population that locates inside the dynamic search boundaries; then, the search space of the algorithm is guided to approximate the space in which the global optimum is included by simultaneously evaluating the current population and the opposite one. This approach takes full advantage of the characteristics of the elite individuals that contains more beneficial search for information compared to the ordinary individuals and guide this algorithm approach to the global optimal solution. On the other hand, the proposed algorithm can greatly enrich the diversity of the population and improve the calculation accuracy, which leads to a good optimization performance. What is more, the new method can enhance the quality of solutions without losing the robustness. The experiments are conducted on 14 classic benchmark functions and 28 more complex functions from the IEEE CEC'2013 competition, and the experimental results compared with the other meta-heuristic algorithms (including four

improved CS algorithms) show that the proposed algorithm is much better than the compared ones at not only the accuracy of solutions but also for the convergence speed and has strong global searching ability and local optimization ability.

Acknowledgments: This work is supported by the National Science Foundation of China under Grants Nos. 61165015, 61463007, 61563008 and the Project of Guangxi University for Nationalities Science Foundation under Grant No. 2012MDZD037.

Bibliography

[1] B. Alatas and E. Akin, Chaotically encoded particle swarm optimization algorithm and its applications, *Chaos Solit. Fract.* **41** (2009), 939–950.
[2] F. S. A1-Qunaieer, H. R. Tizhoosh and S. Rahnamayan, Opposition based computing-a survey, in: *Proceedings of International Joint Conference on Neural Networks*, pp. 1–7, Barcelona, Spain, 2010.
[3] K. Chandrasekaran and S. P. Simon, Multi-objective scheduling problem: hybrid approach using fuzzy assisted cuckoo search algorithm, *Swarm Evol. Comput.* **5** (2012), 1–16.
[4] V. R. Chifu, C. B. Pop, I. Salomie, D. S. Suia and A. N. Niculici, Optimizing the semantic web service composition process using cuckoo search, *Intell. Distributed Comput. V Stud. Computat. Intell.* **382** (2012), 93–102.
[5] K. Choudhary and G. N. Purohit, A new testing approach using cuckoo search to achieve multi-objective genetic algorithm, *J. Comput.* **3** (2011), 117–119.
[6] S. Das, A. Abraham, U. K. Chakraborty and A. Konar, Differential evolution using a neighborhood-based mutation operator, *IEEE Trans. Evol. Comput.* **13** (2009), 526–553.
[7] A. H. Gandomi, X. S. Yang, S. Talatahari and S. Deb, Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization, *Comput. Math. Appl.* **63** (2012), 191–200.
[8] A. H. Gandomi, X. S. Yang and A. H. Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems, *Eng. Comput.* **29** (2013), 17–35. doi:10.1007/s00366-011-0241-y.
[9] S. Garcia, D. Molina, M. Lozano and F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization at CEC-05, Edinburgh, UK, 2–5 September 2005, *J. Heuristics.* 2008, doi:10.1007/s10732-008-9080-4.
[10] J. H. Holland, *Adaptation in natural and artificial systems*, MIT Press, Cambridge, USA, 1992.
[11] D. Karaboga and B. Akay, A comparative study of artificial bee colony algorithm, *Appl. Math. Comput.* **214** (2009), 108–132.
[12] A. Kaveh, *Advances in metaheuristic algorithms for optimal design of structures*, Springer, Switzerland, 2014.
[13] A. Kaveh and T. Bakhshpoori, Optimum design of steel frames using cuckoo search algorithm with Levy flights. Structural design of tall and special buildings, vol 21, online first 28 Nov 2011. http://onlinelibrary.wiley.com/doi/10.1002/tal.754/abstract.
[14] A. Kaveh and T. Bakhshpoori, Optimum design of steel frames using Cuckoo Search algorithm with Lévy flights, *Struct Des Tall Special Buil* **22** (2013), 1023–1036.
[15] A. Kaveh and N. Farhoudi, A new optimization method: dolphin echolocation, *Adv. Eng. Softw.* **59** (2013), 53–70.
[16] A. Kaveh and M. Khayatazad, A new meta-heuristic method: ray optimization, *Comput. Struct.* **112** (2012), 283–294.
[17] A. Kaveh and V. R. Mahdavi, Colliding bodies optimization: a novel meta-heuristic method, *Comput. Struct.* **139** (2014), 18–27.
[18] A. Kaveh and S. Talatahari, A novel heuristic optimization method: charged system search, *Acta Mech.* **213** (2010), 267–289.
[19] J. Kennedy and R. Eberhart, Particle swarm optimization, in: *Proc. of the IEEE International Conference on Neural Networks*, IV: 1942–1948, Perth, Australia, 1995.
[20] A. Kumar and S. Chakarverty, Design optimization for reliable embedded system using Cuckoo search, in: *Proceedings of 3rd International Conference on Electronics Computer Technology* (*ICECT2011*), pp. 564–568, 2011.
[21] J. J. Liang, B. Y. Qu, P. N. Suganthan and A. G. Hernández-Díaz, Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212 (2013).
[22] A. T. Mehrabian and C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, *Ecol. Inform.* **1** (2006), 355–366.
[23] A. Mucherino and O. Seref, Monkey search: a novel metaheuristic search for global optimization, in: *Proc. of the American Institute of Physics Conference*, pp. 162–173, Gainesville, USA, 2007.
[24] W. T. Pan, *Fruit fly optimization algorithm*, Tsang Hai Book Publishing Co., Taipei, China, 2011, pp. 10–12 (in Chinese).
[25] I. Pavlyukevich, Lévy flights, non-local search and simulated annealing, *J Comput. Phys.* **226** (2007), 1830–1844.

[26] K. Perumal, J. M. Ungati, G. Kumar, N. Jain, R. Gaurav and P. R. Srivastava, Test data generation: a hybrid approach using cuckoo and tabu search, swarm, evolutionary, and memetic computing (SEMCCO2011), *Lect. Notes Comput. Sci.* **7077** (2011), 46–54.
[27] P. R. Srivastava, M. Chis, S. Deb and X. S. Yang, An efficient optimization algorithm for structural software testing, *Int. J. Artif. Intell.* **9** (2012), 68–77.
[28] R. Storn and K. Price, Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces, *J. Glob. Optim.* **11** (1997), 341–359.
[29] T. Takahama and S. Sakai, Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elite, in: *Proc. of IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, Vancouver, 2006.
[30] L. H. Tein and R. Ramli, Recent advancements of nurse scheduling models and a potential path, in: *Proceedings of 6th IMT-GT Conference on Mathematics, Statistics and Its Applications* (*IC-MSA 2010*), pp. 395–409, 2010.
[31] R. A. Vazquez, Training spiking neural models using cuckoo search algorithm, in: *2011 IEEE Congress on Evolutionary Computation* (*CEC'11*), pp. 679–686, 2011.
[32] S. Walton, O. Hassan, K. Morgan and M. R. Brown, Modified cuckoo search: a new gradient free optimization algorithm, *Chaos Solitons Fractals* **44** (2011), 710–718.
[33] G. G. Wang, L. H. Guo, H. Duan, H. Q. Wang, L. Liu and M. Z. Shao, A hybrid meta-heuristic DE/CS algorithm for UCAV three-dimension path planning, *Scientific World Journal* **2012** (2012), Article ID 583973, 11 pages.
[34] L. J. Wang, Y. L. Yin and Y. W. Zhong Cuckoo search algorithm with dimension by dimension improvement, *J. Softw.* **24** (2013), 2687–2698.
[35] L. Wei-gao, O. Xu and L. De-xiang, Two dimensional numerical integration based on chaotic cuckoo search optimization algorithm, *Microelectron. Comput.* **31** (2014), 149–154.
[36] Y. Xinshe, Multiobjective firefly algorithm for continuous optimization, *Eng. Comput.* **29** (2013), 175–184.
[37] X. S. Yang, A new meta-heuristic bat-inspired algorithm, in: *Proceedings of the International Workshop on Nature Inspired Cooperative Strategies for Optimization* (*NICSO'10*), Volume 284 of the series Studies in Computational Intelligence, pp. 65–74, Springer, Berlin, Heidelberg, 2010.
[38] X. S. Yang and S. Deb, Cuckoo search via Lévy flights, in: *Proceedings of world congress on nature and biologically inspired computing* (*NaBIC 2009*), pp. 210–214, IEEE Publications, USA, 2009.
[39] X. S. Yang and S. Deb, Engineering optimization by cuckoo search, *Int. J. Math. Modell. Num. Opt.* **1** (2010), 330–343.
[40] X. S. Yang and S. Deb, Multiobjective cuckoo search for design optimization, *Comput. Oper. Res.* **40** (2013), 1616–1624.
[41] A. R. Yildiz, Cuckoo search algorithm for the selection of optimal machine parameters in milling operations, *Int. J. Adv. Manuf. Technol.* **64** (2013), 55–61.
[42] H. Q. Zheng, and Y. Zhou, A novel cuckoo search optimization algorithm based on Gauss distribution, *J. Comput. Inf. Syst.* **8** (2012), 4193–4200.
[43] X. Zhou, Z. Wu, H. Wan, et al., Elite opposition-based particle swarm optimization, *Acta Electronica Sinica* **41** (2013), 1647–1652.