International Journal of Innovative
Computing, Information and Control
Volume **17**, Number **3**, June **2021**

ICIC International ©2021 ISSN 1349-4198
pp. **887–903**

PARALLEL CUCKOO SEARCH FOR GLOBAL OPTIMIZATION

Supaporn Suwannarongsri

Department of Materials Handling and Logistics Engineering
Faculty of Engineering
King Mongkut's University of Technology North Bangkok
1518 Pracharaj-1 Rd., Bangsue, Bangkok 10800, Thailand
supaporn.s@eng.kmutnb.ac.th

Received December 2020; revised April 2021

Abstract. *The cuckoo search (CS) has become acceptable worldwide as one of the most efficient metaheuristic optimization search techniques applied to various real-world problems. Since the first appearance, many variants of the CS have been developed to improve its search performance. In this paper, the newest modified version of the CS named the parallel cuckoo search (PCS) is proposed for running on a single-CPU platform. Based on the original CS by using the random process drawn from the Lévy distribution, the proposed PCS contains the CS as its search core. In the PCS algorithm, the partitioning strategy (PS) is conducted to divide an entire search space into many sub-search-spaces for each CS. The sequencing strategy (SS) is employed to organize the search units to run one-by-one on a single iteration/generation. Also, the discarding strategy (DS) is applied to discarding some unlikely to be successful CS. To perform its search performance, the proposed PCS is tested against ten selected benchmark optimization problems compared with the original CS. As experimental results, the proposed PCS performs more efficient in global optimization of ten selected benchmark optimization problems with higher success rates, less search generations and less search times than the original CS.*
Keywords: Parallel cuckoo search, Global optimization, Metaheuristic optimization technique

1. **Introduction.** Metaheuristic algorithms have become powerful and popular in computational intelligence and applications to almost all areas of real-world optimization problems [1-3]. Recently, the most interesting metaheuristic algorithms are developed and launched, for example, modified bat algorithm (MBA) [4], multiobjective Lévy-flight firefly algorithm (mLFFA) [5], modified flower pollination algorithm (MoFPA) [6], improved water evaporation optimization (IWEO) [7], hybrid gray wolf optimization and particle swarm optimization (GWOPSO) [8], stingless bee algorithm (SBA) [9] and android-based optimization [10]. Following the literature, the cuckoo search (CS) was firstly proposed by Yang and Deb in 2009 as one of the most efficient nature-inspired metaheuristic algorithms for numerical function optimization and continuous problems [11]. The CS algorithm was developed by its inspiration from the brooding parasitism of cuckoo species in nature and by using the random process drawn from the Lévy distribution. The algorithm of the CS was proved for the global convergent property [12]. Moreover, it was tested many well-known benchmark functions and compared with the genetic algorithm (GA) and particle swarm optimization (PSO), and it was found that the CS performed promising results better than the GA and PSO [11,13,14]. Since then, the CS has been applied in almost all areas [15,16], such as function optimization, engineering optimization, image processing, scheduling, planning, feature selection, forecasting and other real-world applications. In

DOI: 10.24507/ijicic.17.03.887

addition, many variants of the CS algorithm have been developed by many researchers to improve its search performance [16]. The significant variant of the CS algorithm includes the discrete binary CS (BCS) algorithm developed to get binary solutions of binary real-world optimization problems, i.e., knapsack problems [17] and traveling salesman problems [18]. The neural-swarm CS (NSCS) is one of the significant variants of the CS algorithm developed by combining the artificial neural network (ANN) and CS algorithm for health and safety diagnosis [19]. The quantum-inspired CS (QCS) was a new framework of hybridization between quantum-inspired computing and bio-inspired computing proposed for solving knapsack problems [20]. The emotional chaotic CS (ECCS) was developed based on the Lévy distribution of the CS and the psychology model of emotion and chaotic sequence to enrich the searching behavior and avoid being trapped by local optima of the reconstruction of chaotic Lorenz dynamic system [21]. The modified CS (MCS) is another variant of the CS algorithm developed by adjusting the step size determined from the sorted rather than only permuted fitness matrix for solving unconstrained optimization problems [22]. The multiobjective CS was one of the most efficiently variants of the CS algorithm developed to solve both standard and real-world multiobjective optimization problems including welded beam design and disc brake design [14]. The hybrid CS/GA was proposed by using the crossover strategy of GA in order to lay more eggs of CS for solving global optimization problems [23]. Also, the hybrid CS/PSO was developed by using the swarm intelligence in PSO in order to reach to better solutions and reduce the chance of the cuckoo eggs to be discovered by the host birds for solving global optimization problems [24].

One of the most interesting variants of the CS algorithm is the parallel CS. From literature review, the parallelized CS was proposed in 2012 for solving the unconstrained optimization problems [25]. The algorithm of the parallelized CS in [25] consisting of many original CS algorithms (CSs) was developed for running on multicore processors. All CS algorithms in [25] were not modified. With their identical algorithms, all CSs in [25] were defined for searching on the same entire search space. By this concept, it does not differ from the parallel personal computer (PC) in which the original CS was installed. Moreover, the parallelized CS in [25] needs high cost processor. Development of the parallelized CS for running on a single-CPU is the challenging task because it is more difficult but cheaper.

In order to improve the search performance of the CS as the parallelized manner for running on a single-CPU, the newest modified version of the CS named the parallel cuckoo search (PCS) is proposed in this paper. By using the original CS as its search core, the proposed PCS contains the partitioning strategy (PS) to divide an entire search space into many sub-search-spaces for each CS, the sequencing strategy (SS) to organize the search units to run one-by-one on each iteration/generation and the discarding strategy (DS) to discard some unlikely to be successful CS. This paper consists of five sections. After an introduction presented in Section 1, the rest of the paper is arranged as follows. The original CS and the proposed PCS algorithm are described in Section 2. Ten selected benchmark optimization problems for global optimization (minimization) are detailed in Section 3. Experimental results and discussions are illustrated in Section 4, while the conclusions are followed in Section 5.

2. **Parallel Cuckoo Search Algorithm.**

The algorithm of the original CS is briefly described in this section. Then, the algorithm of the proposed PCS is elaborately given as follows.

2.1. **Original CS algorithm.** The original CS algorithm mimics the obligate brood parasitic behavior of some cuckoo species in nature by combination with the Lévy flight behavior of some birds and fruit flies [11,26]. The algorithm of the original CS can be represented by the flow diagram as visualized in Figure 1, where $f(\boldsymbol{x})$ is the objective function to be minimized, $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ is the solution vector in d dimensions, $\boldsymbol{x}^*$ is an initial solution, Max_Gen is the maximum generation, Gen is the generation counter, n is number of cuckoos and m is number of cuckoo's eggs that were found by the host bird. With the iteration process, n cuckoos will find the new nests by a random process drawn from Lévy flight distribution and lay their eggs in the random nests to create new solutions. After cuckoos lay their eggs, the host bird will check their nests with a fraction $p_a \in [0, 1]$. If the host bird discovers m ($m \leq n$) cuckoo eggs, it will either throw these m eggs away or simply abandon its nest and build a new nest. Then, m cuckoos need to find the new nests by a random process drawn from Lévy flight distribution again and lay their eggs in the new random nests to create new solutions. After that, all cuckoo's eggs (new solutions) will be evaluated by the objective function $f(\boldsymbol{x})$ and the current best solution will be updated by the better solution found. The CS algorithm will be iteratively processed until the optimal solution is found or the termination criteria (TC) are met. Generally, the TC will be set from the difference between Gen and Max_Gen. If Gen $\leq$ Max_Gen, the search process will continue. Otherwise the search process will be stopped.

In the original CS algorithm, new solutions $\boldsymbol{x}^{(t+1)}$ for cuckoo i can be calculated by the current solutions $\boldsymbol{x}^{(t)}$ for cuckoo i associated with a random process drawn from Lévy flight distribution as stated in (1), where $\alpha > 0$ is the step size (in most cases, $\alpha = 1$), $\oplus$ means entrywise multiplication and Lévy(λ) represents a random process drawn from Lévy flight distribution having an infinite variance with an infinite mean as expressed in (2), where λ is the mean or expectation of the occurrence of the event during a unit interval and β is an index of the Lévy distribution. The step length s of cuckoo flight can be calculated by (3), where u and v are random processes drawn from normal distribution as stated in (4). Standard deviations σ_u and σ_v of u and v are also expressed in (5), where Γ is the standard Gamma function [11-14].

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda) \tag{1}$$

$$\text{Lévy} \approx u = t^{-\lambda} = t^{-1-\beta}, \quad (1 < \lambda < 3), \quad (0 < \beta \leq 2) \tag{2}$$

$$s = \frac{u}{|v|^{1/\beta}} \tag{3}$$

$$u \approx N\left(0, \sigma_u^2\right), \quad v \approx N\left(0, \sigma_v^2\right) \tag{4}$$

$$\sigma_u = \sqrt[\beta]{\frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}}}, \quad \sigma_v = 1 \tag{5}$$

2.2. **Proposed PCS algorithm.** In order to improve the search performance of the original CS, the PCS is proposed by utilizing the exploration strategies. The PCS algorithm proposed in this paper differs from the existing parallelized CS algorithm [25] in that the PCS is developed for running on a single-CPU platform, while the parallelized CS [25] was developed for running on multicore processors. By using the original CS algorithm shown in Figure 1 as its search core, the proposed PCS algorithm consists of partitioning strategy (PS), sequencing strategy (SS) and discarding strategy (DS), respectively.

2.2.1. *Partitioning strategy (PS).* The PS operates only once at the beginning of the search to divide an entire search space into many sub-search-spaces. Symmetrical partitioning can be easily made, whereas asymmetrical partitioning can be employed. However, too many sub-search-spaces would result in a slow search process. The PS also defines

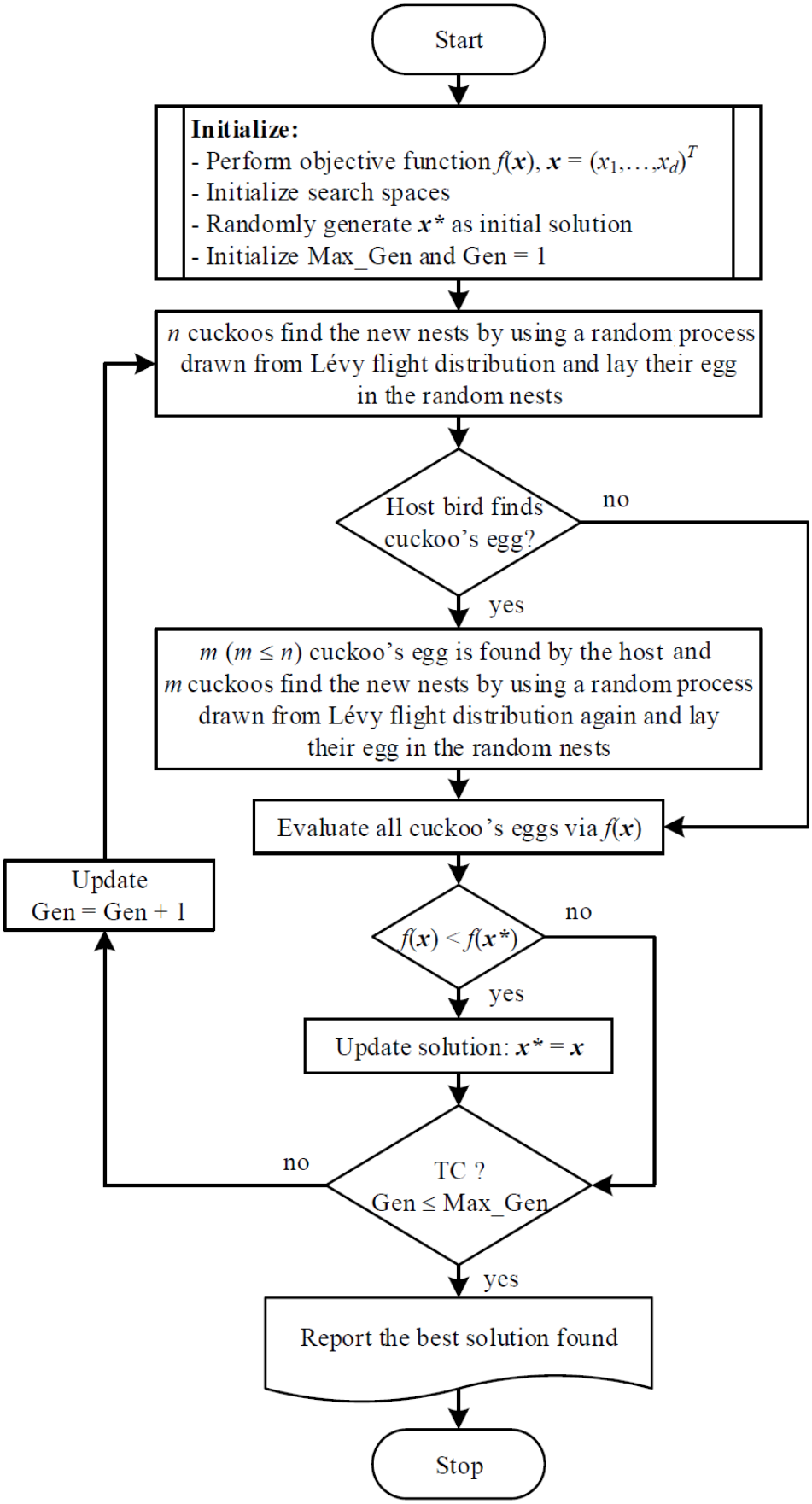


FIGURE 1. Flow diagram of CS algorithm

explicit boundaries for the sub-search-spaces, each of which is given an individual initial solution for an independent CS. All CSs as the search cores of the PCS use the same set of search parameters. Just before launching the search, the PS removes the predefined boundaries such that all CSs could search freely on the originally entire search space. This strategy helps to reduce any conflicts that may arise during the search along the border lines. Figure 2 summarizes the PS procedures.

Step_PS1: Load number of CSs, N, as the search core of the PCS, and original entire search space.
Step_PS2: Partition the entire search space into N sub-search-spaces.
Step_PS3: Randomly generate initial solutions for each sub-search-space, and remove partitions.

FIGURE 2. PS procedures

2.2.2. *Sequencing strategy (SS).* The SS is a time-sharing tactic to organize all CSs to run one-by-one on a single iteration/generation approach providing the hardware has a single-CPU. With the SS, all CSs in the PCS will be run one-by-one as sequential manner on each iteration. The operation is repeated until one of the CSs hits the solution, and requests for exiting the search according to the satisfied termination criterion (TC). Either an equal or an unequal number of search rounds constitutes one search trial. The works by this paper utilize an equal number of search rounds for all CSs to work on one search trial. The SS will be interrupted by the DS for some information transferred between SS and DS. With the DS, as time goes by, more and more CSs will be cut off from the search process until only one CS is left to continue searching. However, an individual CS can request for a complete termination once the global solution is found at any time. The SS needs to communicate with the DS to learn about the existing and the forcedly terminated some CSs. Thus, the SS can be summarized by the procedures shown in Figure 3.

Step_SS1: Communicate with DS to receive information concerning the discarded CSs.
Step_SS2: Launch all existing CSs one-by-one for each iteration in a sequential manner.
Step_SS3: If the TC is met, exit the search, otherwise go back to Step_SS1.

FIGURE 3. SS procedures

2.2.3. *Discarding strategy (DS).* The DS serves to discard some unlikely to be successful CS to reduce the overall search time of the PCS. Various possible approaches can be used to implement this idea. In this paper, a simple implementation by using the evaluation of the cost values of the current best solutions of the CSs is demonstrated associated with the number of iterations/generations. In each time of the DS operates, the number of CSs is reduced by half. After this forced termination made to the low-quality CSs, the DS transfers the information concerning the being-terminated and the existing CSs to the SS. Figure 4 summarizes the DS procedures.

By using the CS as the search core, the PCS procedures can be summarized in Figure 5. It can be noticed that during Step_PCS3 the CSs as search cores work independently without any modifications or intrusion. Hence, the convergent property of the CSs is

Step_DS1: Load cost values of the current best solutions of all CSs.
Step_DS2: Sort min-max values of all cost values.
Step_DS3: Keep CSs from the top to the middle of the sorted list as active, discard the rest of CSs.
Step_DS4: Transfer information in Step_DS3 to the SS.

FIGURE 4. DS procedures

Step_PCS1: Identify entire search spaces, define number of search cores, initializing N search-unit algorithms (CSs), Max_Gen and Gen = 1.
Step_PCS2: Activate the PS, situate CSs to sub-search-spaces, and remove partitions.
Step_PCS3: Invoke the SS and execute CS_1, CS_2,..., CS_N, for $N = N - K$, $K \leq N - 1$, $N_{min} = 1$.
Step_PCS4: If the TC is met (Gen ≤ Max_Gen), exit with the global solution.
Step_PCS5: Activate the DM.
Step_PCS6: Update Gen = Gen + 1, and go back to Step_PCS2.

FIGURE 5. PCS procedures

always preserved. The flow diagram of the PCS algorithm can be represented by the flow diagram as shown in Figure 6, where $CS_1, CS_2, \ldots, CS_N$ are the CS algorithms represented in Figure 1.

3. **Benchmark Optimization Problems.** For global optimization by the proposed PCS algorithm, ten well-known surface optimization problems are selected as the benchmark functions, i.e., Bohachevsky function (BF), De Jong function (DJF), Griewank function (GF), Michaelwicz function (MF), Rastrigin function (RF), Salomon function (SalF), Schwefel function (SchF), Sinusoid function (SiF), Shekel's fox-holes function (SF) and Yang function (YF) [27,28]. These benchmark functions are selected from the difference of their characteristics including linearity, symmetry and modality (basins and valleys). Details of these selected functions are summarized in Table 1.

4. **Experimental Results and Discussions.** To perform its search performance, the proposed PCS algorithm will be tested against ten selected benchmark functions as detailed in the previous section, i.e., BF, DJF, GF, MF, RF, SalF, SchF, SiF, SF and YF. Performance comparisons are made among the original CS and the proposed PCS with 2, 4 and 8 CSs denoted as PCS#2, PCS#4 and PCS#8, respectively. Both the original CS and the proposed PCS algorithms were coded by MATLAB version 2018b run on the Intel(R) Core(TM) i5-3470 CPU@3.60GHz, 4.0GB-RAM. The searching parameters of all CSs are set by the recommendations of Yang and Deb [11,12] as the same values, i.e., number of cuckoos $n = 40$ and fraction $p_a = 0.2$ (20%). 100 trials are conducted to find the best solution of each function. Both the original CS and the proposed PCS will be terminated once two termination criteria (TC) are satisfied, i.e., 1) the function values are less than a given tolerance $\varepsilon \leq 10^{-6}$ from f_{min} of each function shown in Table 1 or 2) the search meets the maximum generation (Max_Gen = 2,000). The former criterion implies that the search is successful, while the later means that the search is not successful. For the PCS algorithm, the PS, SS and DS strategies are set as follows.

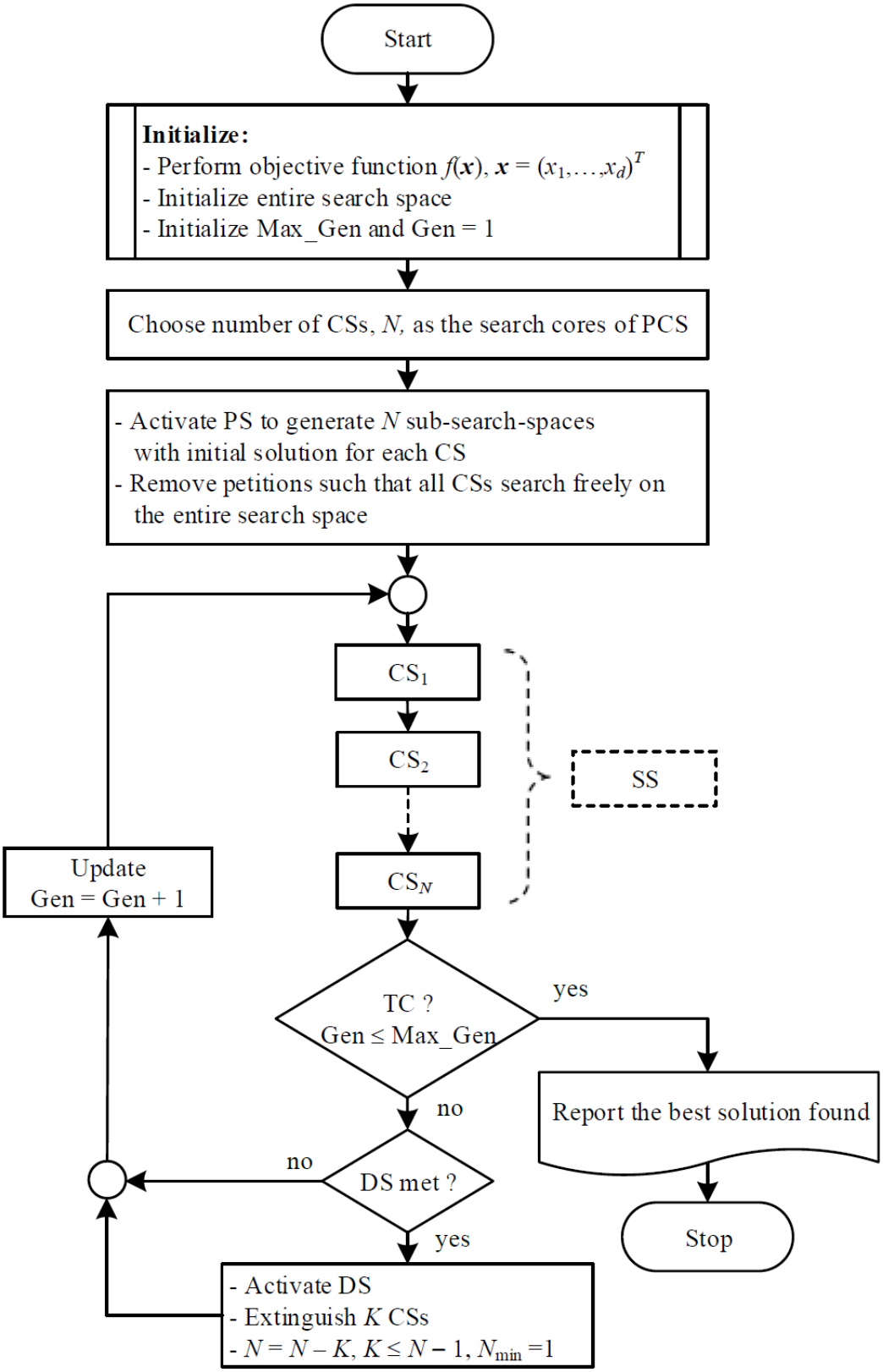
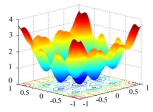
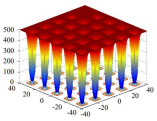
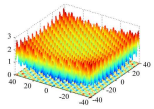
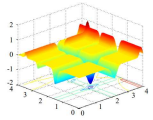
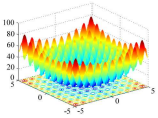
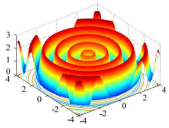
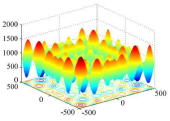
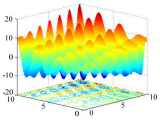
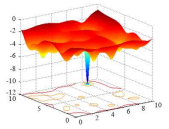
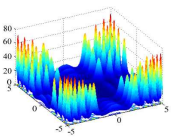


FIGURE 6. Flow diagram of the proposed PCS algorithm

TABLE 1. Ten selected benchmark functions

Names	Functions/ Minimum solutions	Search spaces	Sketches
Bohachevsky function (BF)	$f_1(x,y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$ $f_{\min}(0,0) = 0$	$[-100, 100]$	
De Jong function (DJF)	$f_2(x,y) = \left[1/500 + \sum_{j=1}^{25} \left\{1/\left(j + (x-a_{1j})^6 + (y-a_{2j})^6\right)\right\}\right]^{-1}$ $a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$ $f_{\min}(-32,-32) = 0.9980$	$[-40, 40]$	
Griewank function (GF)	$f_3(x,y) = \frac{1}{4000}(x^2+y^2) - \cos\left(\frac{x}{\sqrt{2}}\right)\cos\left(\frac{y}{\sqrt{2}}\right) + 1$ $f_{\min}(0,0) = 0$	$[-100, 100]$	
Michaelwicz function (MF)	$f_4(x,y) = -\sin(x)\sin(x^2/\pi)^{20} - \sin(y)\sin(2y^2/\pi)^{20}$ $f_{\min}(2.20319, 1.57049) = -1.8013$	$[0, \pi]$	
Rastrigin function (RF)	$f_5(x,y) = 20 + x^2 + y^2 - 10[\cos(2\pi x) + \cos(2\pi y)]$ $f_{\min}(0,0) = 0$	$[-5.12, 5.12]$	
Salomon function (SalF)	$f_6(x,y) = 1 - \cos\left(2\pi\sqrt{x^2+y^2}\right) - 0.1\sqrt{x^2+y^2}$ $f_{\min}(0,0) = 0$	$[-100, 100]$	
Schwefel function (SchF)	$f_7(x,y) = 837.9658 - \left(x\sin\sqrt{\lvert x\rvert} + y\sin\sqrt{\lvert y\rvert}\right)$ $f_{\min}(420.9687, 420.9687) = 0$	$[-500, 500]$	
Sinusoid function (SiF)	$f_8(x,y) = x\sin(4x) + 1.1y\sin(2y)$ $f_{\min}(9.039, 8.668) = -18.5547$	$[0, 10]$	
Shekel's fox-holes function (SF)	$f_9(x,y) = -\sum_{j=1}^{30}\left[1/\left\{c_j + (x-a_{1j})^2 + (y-a_{2j})^2\right\}\right]$ $a_{ij} = \begin{pmatrix} 9.6810 & 9.4000 & 8.0250 & \cdots & 9.4960 & 4.1380 \\ 0.6670 & 2.0410 & 9.1520 & \cdots & 4.8300 & 2.5620 \end{pmatrix}$ $c_j = (0.806\ 0.517\ 0.100\ 0.908\ \cdots\ 0.608\ 0.326)$ $f_{\min}(8.0241, 9.1465) = -12.1190$	$[0, 10]$	
Yang function (YF)	$f_{10}(x,y) = (\lvert x\rvert + \lvert y\rvert)\exp\left[-\left(\sin x^2 + \sin y^2\right)\right]$ $f_{\min}(0,0) = 0$	$[-2\pi, 2\pi]$	

The PS settings are explained by referring to Table 2, which declares the symmetrical boundaries of the sub-search-spaces for 2, 4 and 8 CSs for each function. For example, let us consider the BF having its entire search space defined by $[x_{\max}\ y_{\max}; x_{\min}\ y_{\min}] = [100\ 100; -100\ -100]$. For the PCS#4 consisting of 4 CSs, the BF is decomposed into #1$[0\ 100; -100\ 0]$ for the 1st sub-search-space, #2$[100\ 100; 0\ 0]$ for the 2nd sub-search-space, #3$[0\ 0; -100\ -100]$ for the 3rd sub-search-space and #4$[100\ 0; 0\ -100]$ for the 4th sub-search-space, respectively. The similar approach for search space partitioning can be applied to the other functions. In addition, other geometric partitioning techniques and coordinates could be employed to suit particular applications.

TABLE 2. Sub-search-spaces partitioned by PS

Functions	**Entire search spaces**	**Sub-search-spaces defined by PS**		
		PCS#2	**PCS#4**	**PCS#8**
BF, GF and SalF	[100 100; –100 –100]	#1[0 100; –100 –100] #2[100 100; 0 –100]	#1[0 100; –100 0] #2[100 100; 0 0] #3[0 0; –100 –100] #4[100 0; 0 –100]	#1[–50 100; –100 0] #2[0 100; –50 0] #3[50 100; 0 0] #4[100 100; 50 0] #5[–50 0; –100 –100] #6[0 0; –50 –100] #7[50 0; 0 –100] #8[100 0; 50 –100]
DJF	[40 40; –40 –40]	#1[0 40; –40 –40] #2[40 40; 0 –40]	#1[0 40; –40 0] #2[40 40; 0 0] #3[0 0; –40 –40] #4[40 0; 0 –40]	#1[–20 40; –40 0] #2[0 40; –20 0] #3[20 40; 0 0] #4[40 40; 20 0] #5[–20 0; –40 –40] #6[0 0; –20 –40] #7[20 0; 0 –40] #8[40 0; 20 –40]
MF	[π π; 0 0]	#1[π/2 π; 0 0] #2[π π; π/2 0]	#1[π/2 π; 0 π/2] #2[π π; π/2 π/2] #3[π/2 π/2; 0 0] #4[π π/2; π/2 0]	#1[π/4 π; 0 π/2] #2[π/2 π; π/4 π/2] #3[3π/4 π; π/2 π/2] #4[π π; 3π/4 π/2] #5[π/4 π/2; 0 0] #6[π/2 π/2; π/4 0] #7[3π/4 π/2; π/2 0] #8[π π/2; 3π/4 0]
RF	[5.12 5.12; –5.12 –5.12]	#1[0 5.12; –5.12 –5.12] #2[5.12 5.12; 0 –5.12]	#1[0 5.12; –5.12 0] #2[5.12 5.12; 0 0] #3[0 0; –5.12 –5.12] #4[5.12 0; 0 –5.12]	#1[–2.56 5.12; –5.12 0] #2[0 5.12; –2.56 0] #3[2.56 5.12; 0 0] #4[5.12 5.12; 2.56 0] #5[–2.56 0; –5.12 –5.12] #6[0 0; –2.56 –5.12] #7[2.56 0; 0 –5.12] #8[5.12 0; 2.56 –5.12]
SchF	[500 500; –500 –500]	#1[0 500; –500 –500] #2[500 500; 0 –500]	#1[0 500; –500 0] #2[500 500; 0 0] #3[0 0; –500 –500] #4[500 0; 0 –500]	#1[–250 500; –500 0] #2[0 500; –250 0] #3[250 500; 0 0] #4[500 500; 250 0] #5[–250 0; –500 –500] #6[0 0; –250 –500] #7[250 0; 0 –500] #8[500 0; 250 –500]
SiF and SF	[10 10; 0 0]	#1[5 10; 0 0] #2[10 10; 5 0]	#1[5 10; 0 5] #2[10 10; 5 5] #3[5 5; 0 0] #4[10 5; 5 0]	#1[2.5 10; 0 5] #2[5 10; 2.5 5] #3[7.5 10; 5 5] #4[10 10; 7.5 5] #5[2.5 5; 0 0] #6[5 5; 2.5 0] #7[7.5 5; 5 0] #8[10 5; 7.5 0]
YF	[2π 2π; –2π –2π]	#1[0 2π; –2π –2π] #2[π π; π/2 0]	#1[0 2π; –2π 0] #2[2π 2π; 0 0] #3[0 0; –2π –2π] #4[2π 0; 0 –2π]	#1[–π 2π; –2π 0] #2[0 2π; –π 0] #3[π 2π; 0 0] #4[2π 2π; π 0] #5[–π 0; –2π –2π] #6[0 0; –π –2π] #7[π 0; 0 –2π] #8[2π 0; π –2π]

The SS is simple set by a time-sharing technique for all CSs running one-by-one as sequential manner in each generation based on a single-CPU. For example, let us consider the PCS#4 consisting of 4 CSs. In the 1st generation, the CS1 will be run, while CS2, CS3 and CS4 are in waiting state. Once the CS1 finishes its process, it goes to the waiting state. At this time, the CS2 will be run, while the CS1, CS3 and CS4 are in waiting state. The operation goes on in this manner until the CS4 finishes its process. Then, the 1st generation is finished. Afterward, the CPU then returns to start the 2nd generation by running CS1, CS2, CS3 and CS4 in sequential manner again. The operation is repeated until one of the CSs hits the optimal solution or some CSs are discarded by the DS.

The DS settings are detailed by referring to Table 3 explaining the qth generation at which the DS is invoked and the number of discarded CSs for all functions. In each time of the DS operates, the number of CSs is discarded by half. The number of the CSs being discarded can be set in the manner of a gradual cut, a moderate cut or a sudden cut, depending on the problem of interests. In this paper, the moderate cut is simply employed. For example in Table 3, let us consider the PCS#8. The DS will be activated three times. The DS becomes active firstly at the 500th generation, and 4 low-quality CSs are discarded. The DM becomes active secondly and thirdly at the 1000th and 1500th generations; 2 and 1 CSs are forced to stop, respectively. Eventually, there is only one CS left to continue searching.

Table 3. DS settings

For all functions	**The qth generation at which the DS is invoked and the number of discarded CSs**					
	PCS#2	**PCS#4**		**PCS#8**		
	1st	**1st**	**2nd**	**1st**	**2nd**	**3rd**
qth generation	1,000	700	1,400	500	1,000	1,500
No. of discarded CSs	1	2	1	4	2	1

Figure 7 shows the convergent rates of the original CS and the proposed PCS#2, PCS#4 and PCS#8 over the BF (1 trial) as an example to demonstrate the DS settings and the behavior of the search process of the original CS and the proposed PCS. Then, Figure 8 shows the convergent rates of the original CS and the proposed PCS#2, PCS#4 and PCS#8 over the BF (100 trials) as an example. The convergent rates of other functions are omitted because they have a similar form to those in Figure 8. Referring to Figure 8, it can be observed that the PCS#8 in Figure 8(d) performs the better search performance with the less average-search generation than the PCS#4 in Figure 8(c), the PCS#2 in Figure 8(b) and the original CS in Figure 8(a), respectively.

Tables 4-9 reveal the obtained results over all functions in 100 trials. The percentages of the success rate (PSR) of the original CS and the proposed PCS are summarized in Table 4. Referring to Table 4, it was found that the proposed PCS can increase the PSR, satisfactorily. The average values of PSR tabulated in Table 4 also confirm this. The numeric data in Table 4 are converted into percentage increase of SR (PISR) of the proposed PCS with-respect-to the original CS by using the following relation stated in (6) for comparison purposes as summarized in Table 5. The numeric data in Table 5 are displayed as bar graphs in Figure 9 to give a clear view of the merits of the proposed PCS over ten selected benchmark functions. From Table 5 and Figure 9, it can be noticed that the PCS#2, PCS#4 and PCS#8 can averagely increase the PSR by 15.96%, 39.61% and 50.30%, respectively, when compared with the original CS. This implies that the more

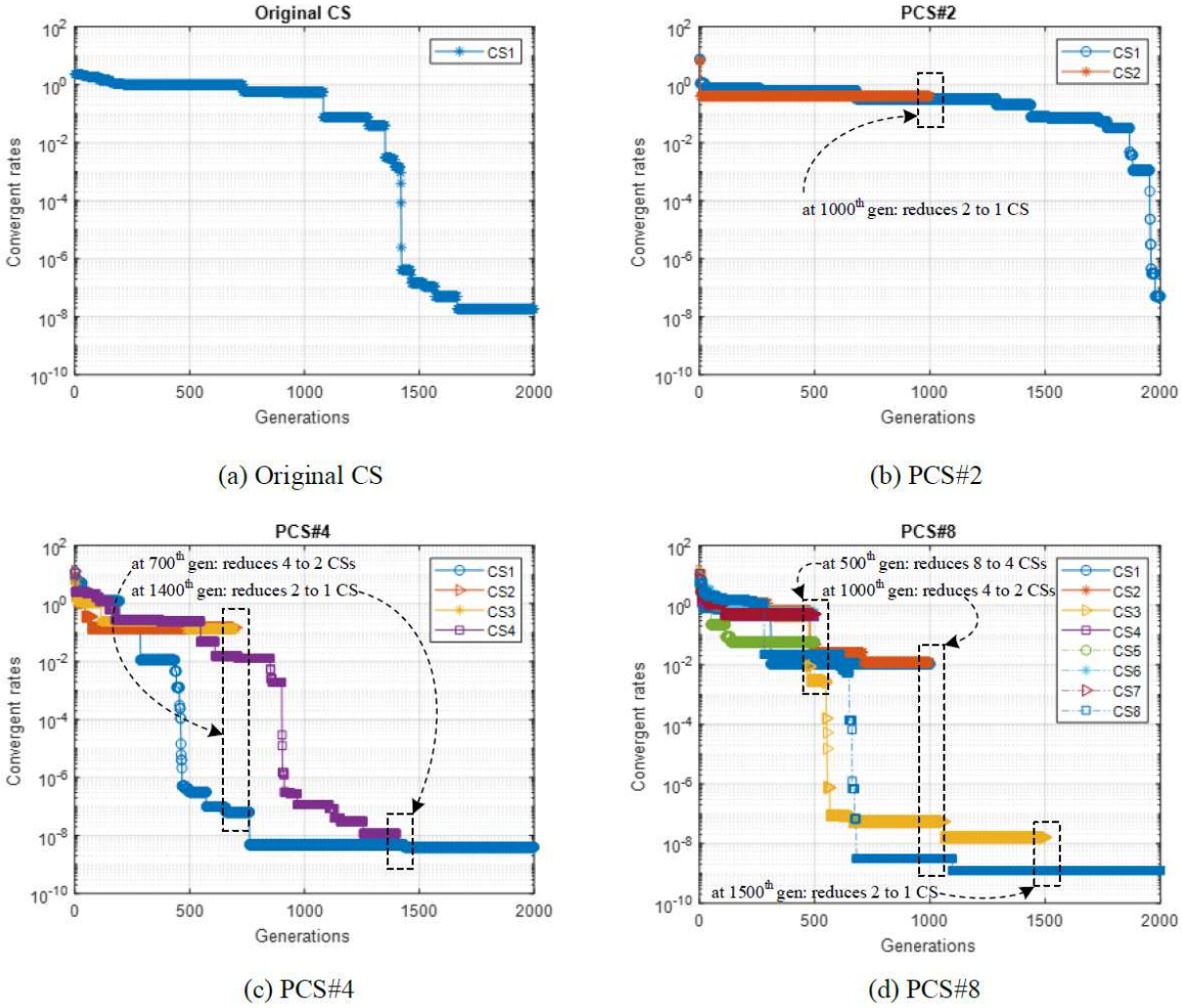


FIGURE 7. Convergent rates of the original CS and the proposed PCS over BF (1 trial)

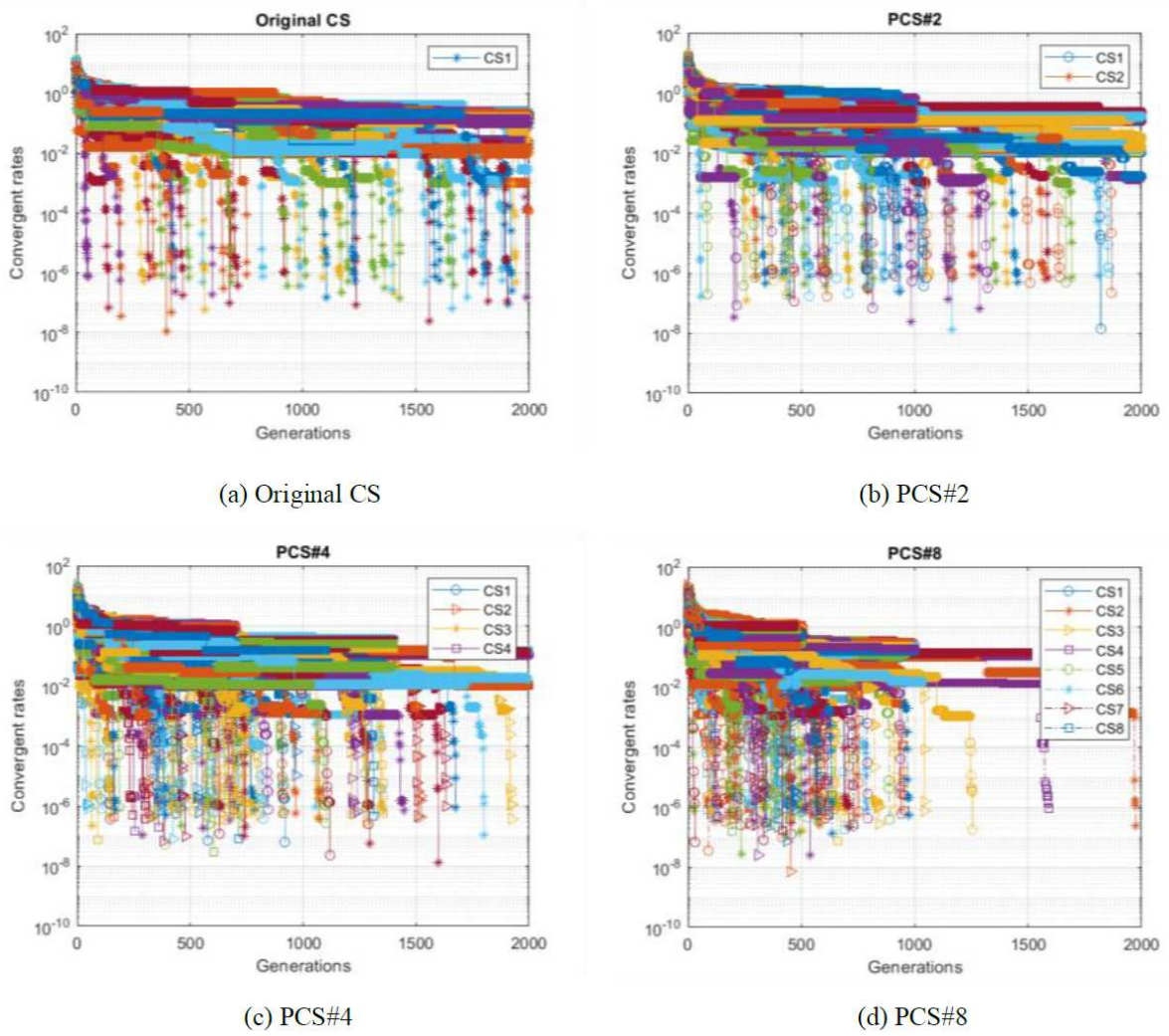
the number of CSs conducted in the PCS, the more the value of PISR.

$$\mathrm{PISR}_{\mathrm{PCS}} = 100 \times \left(\frac{\mathrm{PSR}_{\mathrm{PCS}} - \mathrm{PSR}_{\mathrm{CS}}}{\mathrm{PSR}_{\mathrm{CS}}} \right) \tag{6}$$

Table 6 summarizes the average search generations (SG) of the original CS and the proposed PCS over all functions in 100 trials. Referring to Table 6, it was found that the proposed PCS consumes the SG less than the original CS. The numeric data in Table 6 are transformed into percentage decrease of SG (PDSG) of the proposed PCS with-respect-to the original CS by using the relation stated in (7) for comparison purposes as summarized in Table 7. From Table 7, the PCS#2, PCS#4 and PCS#8 can averagely decrease the SG by 18.39%, 37.83% and 63.05%, respectively, once compared with the original CS. This implies that the more the number of CSs employed in the PCS, the more the value of PDSG.

$$\mathrm{PDSG}_{\mathrm{PCS}} = 100 \times \left(\frac{\mathrm{SG}_{\mathrm{CS}} - \mathrm{SG}_{\mathrm{PCS}}}{\mathrm{SG}_{\mathrm{CS}}} \right) \tag{7}$$

The average search times (ST) consumed by the original CS and the proposed PCS over all functions in 100 trials are summarized in Table 8. It was found that the proposed PCS consumes the ST greater than the original CS. This is because the PCS is proposed for



(a) Original CS (b) PCS#2

(c) PCS#4 (d) PCS#8

FIGURE 8. Convergent rates of the original CS and the proposed PCS over BF (100 trials)

TABLE 4. PSR of the original CS and the proposed PCS

PSR (%) of the original CS and the proposed PCS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	61	72	91	100
DJF	64	82	100	100
GF	53	68	89	99
MF	75	87	100	100
RF	77	89	100	100
SalF	72	93	100	100
SchF	55	66	87	100
SiF	74	83	95	99
SF	71	91	100	100
YF	62	81	98	100
Averages	66.40%	77.00%	92.70%	99.80%

TABLE 5. PISR of the proposed PCS with-respect-to the original CS

PISR (%) of the proposed PCS with-respect-to the original CS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	0.0000	18.0328	49.1803	63.9344
DJF	0.0000	28.1250	56.2500	56.2500
GF	0.0000	28.3019	67.9245	86.7925
MF	0.0000	16.0000	33.3333	33.3333
RF	0.0000	15.5844	29.8701	29.8701
SalF	0.0000	29.1667	38.8889	38.8889
SchF	0.0000	20.0000	58.1818	81.8182
SiF	0.0000	12.1622	28.3784	33.7838
SF	0.0000	28.1690	40.8451	40.8451
YF	0.0000	30.6452	58.0645	61.2903
Averages	0.0000%	15.9639%	39.6084%	50.3012%

TABLE 6. Average search generations (SG) of the original CS and the proposed PCS

Averages search generations (SG) of the original CS and the proposed PCS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	1422.18	1179.12	843.41	488.92
DJF	2568.73	2133.37	1786.60	1103.44
GF	4856.24	4207.36	3028.77	1410.28
MF	1675.37	1350.24	1056.99	778.50
RF	1977.39	1562.43	1180.20	699.31
SalF	2209.06	1978.34	1394.71	824.65
SchF	1818.56	1476.02	1025.19	642.04
SiF	2760.12	2231.89	1857.59	1086.75
SF	1558.28	1209.56	998.87	549.16
YF	1764.23	1317.33	1002.78	601.23
Averages	2261.07	1864.57	1417.51	818.43

use on a single-CPU platform. The ST of the PCS can be reduced by implementation for running on multicore processors. To prove this, the numeric data in Table 8 are converted into the equivalent averages of the ST (EAST) with-respect-to the original CS by using the relation stated in (8). The percentage decrease of ST (PDST) of the equivalent PCS with-respect-to the original CS by using the relation stated in (9) for comparison purposes is detailed in Table 9. It can be noticed that the equivalent PCS#2, PCS#4 and PCS#8 averagely consume the ST less than the original CS by 21.29%, 48.25% and 68.80%, respectively. This implies that the more the number of CSs utilized in the PCS, the more the value of PDST.

$$\mathrm{EAST}_{\mathrm{PCS}\#N} = \frac{\mathrm{ST}_{\mathrm{PCS}\#N}}{N} \tag{8}$$

$$\mathrm{PDST}_{\mathrm{PCS}} = 100 \times \left(\frac{\mathrm{ST}_{\mathrm{CS}} - \mathrm{ST}_{\mathrm{PCS}}}{\mathrm{ST}_{\mathrm{CS}}} \right) \tag{9}$$

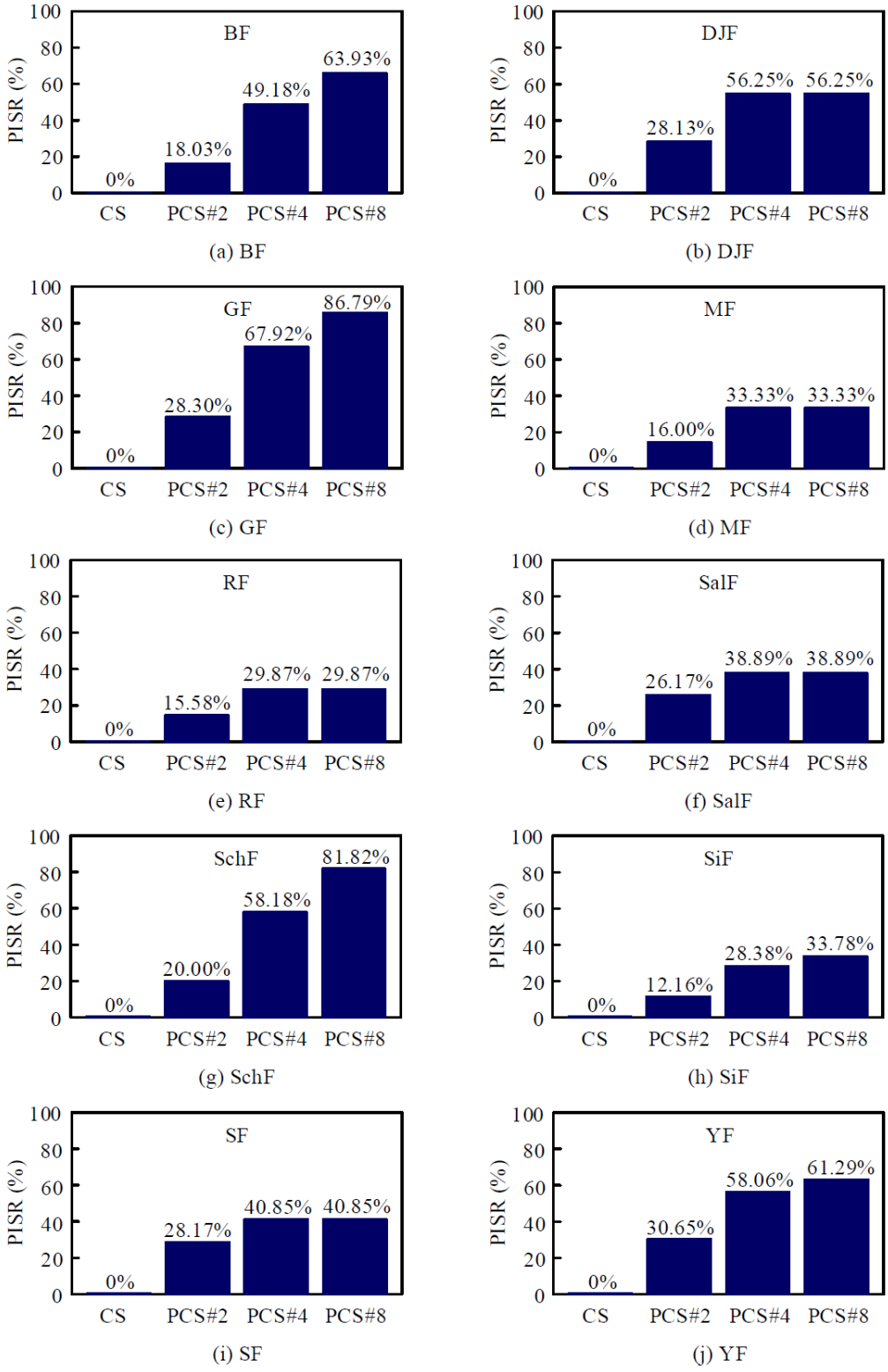


FIGURE 9. Bar graphs of PISR of the proposed PCS with-respect-to the original CS

From Tables 5, 7 and 9, it was found that the proposed PCS algorithm performs more efficient in global optimization of ten selected benchmark optimization problems than the original CS algorithm. As experimental results, the more the number of CSs conducted in the PCS, the higher the search performance for global optimization. As defined earlier,

TABLE 7. PDSG of the proposed PCS with-respect-to the original CS

PDSG (%) of the proposed PCS with-respect-to the original CS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	0.0000	17.0907	40.6960	65.6218
DJF	0.0000	16.9485	30.4481	57.0434
GF	0.0000	13.3618	37.6314	70.9594
MF	0.0000	19.4065	36.9101	53.5327
RF	0.0000	20.9852	40.3153	64.6347
SalF	0.0000	10.4443	36.8641	62.6696
SchF	0.0000	18.8358	43.6263	64.6951
SiF	0.0000	19.1379	32.6989	60.6267
SF	0.0000	22.3785	35.8992	64.7586
YF	0.0000	25.3312	43.1605	65.9211
Averages	0.0000%	18.3920%	37.8250%	63.0463%

TABLE 8. Averages search times (ST) of the original CS and the proposed PCS

Averages ST (sec.) of the original CS and the proposed PCS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	3.6430×10^{-1}	5.0048×10^{-1}	6.7639×10^{-1}	8.9897×10^{-1}
DJF	5.6572×10^{-1}	7.4429×10^{-1}	10.2017×10^{-1}	13.3450×10^{-1}
GF	8.9217×10^{-1}	10.7652×10^{-1}	18.0224×10^{-1}	21.2382×10^{-1}
MF	4.1022×10^{-1}	7.6761×10^{-1}	9.8977×10^{-1}	11.0021×10^{-1}
RF	4.3238×10^{-1}	7.8905×10^{-1}	10.0281×10^{-1}	13.0426×10^{-1}
SalF	5.2398×10^{-1}	8.2910×10^{-1}	9.9989×10^{-1}	12.0229×10^{-1}
SchF	4.0124×10^{-1}	7.3216×10^{-1}	9.6764×10^{-1}	10.7883×10^{-1}
SiF	5.4513×10^{-1}	8.7023×10^{-1}	9.8993×10^{-1}	11.2929×10^{-1}
SF	3.9974×10^{-1}	7.0235×10^{-1}	9.0102×10^{-1}	9.9094×10^{-1}
YF	4.2045×10^{-1}	7.8894×10^{-1}	9.9967×10^{-1}	12.0405×10^{-1}
Averages	4.9553×10^{-1}	7.8007×10^{-1}	10.3495×10^{-1}	12.3672×10^{-1}

the DS setting with a moderate cut is utilized to discard some unlikely to be successful CSs for all test functions. However, DS setting depends on the problems of interest. During test process, it was found that if the function (problem of interests) possesses a small number of local optima (close to unimodal function), DS setting with a sudden cut is more suitable. On the other hand, if the function has a large number of local optima (close to highly multi-modal function), DS setting with a gradual cut is more suitable.

5. **Conclusions.** The novel modified version of the CS named the PCS has been proposed for global optimization running on a single-CPU platform. By using the original CS as its search core, the proposed PCS consists of three distinguished strategies, i.e., the PS conducted to divide an entire search space into many sub-search-spaces for all CSs, the SS employed to organize the search units to run one-by-one on each iteration/generation and the DS applied to discard some unlikely to be successful CS. The proposed PCS has been tested against ten selected benchmark optimization problems for global minimization compared with the original CS. As results, the proposed PCS performs more efficient with higher success rates, less search generations and less search times, than

TABLE 9. Equivalent averages of the ST (EAST) with-respect-to the original CS

EAST (sec.) of the proposed PCS with-respect-to the original CS				
Functions	**Original CS**	**Proposed PCS**		
		PCS#2	**PCS#4**	**PCS#8**
BF	3.6430×10^{-1}	2.5024×10^{-1}	1.6910×10^{-1}	1.1237×10^{-1}
DJF	5.6572×10^{-1}	3.7215×10^{-1}	2.5504×10^{-1}	1.6681×10^{-1}
GF	8.9217×10^{-1}	5.3826×10^{-1}	4.5056×10^{-1}	2.6548×10^{-1}
MF	4.1022×10^{-1}	3.8381×10^{-1}	2.2444×10^{-1}	1.3753×10^{-1}
RF	4.3238×10^{-1}	3.9453×10^{-1}	2.5070×10^{-1}	1.6303×10^{-1}
SalF	5.2398×10^{-1}	4.1455×10^{-1}	2.4997×10^{-1}	1.5029×10^{-1}
SchF	4.0124×10^{-1}	3.6608×10^{-1}	2.4191×10^{-1}	1.3485×10^{-1}
SiF	5.4513×10^{-1}	4.3512×10^{-1}	2.4748×10^{-1}	1.4116×10^{-1}
SF	3.9974×10^{-1}	3.5118×10^{-1}	2.2526×10^{-1}	1.2387×10^{-1}
YF	4.2045×10^{-1}	3.9457×10^{-1}	2.4992×10^{-1}	1.5051×10^{-1}
Averages	4.9553×10^{-1}	3.9004×10^{-1}	2.5644×10^{-1}	1.5459×10^{-1}
PDST (%)	0.00%	21.2883%	48.2493%	68.8031%

the original CS. It can be concluded that the proposed PCS is one of the most efficient metaheuristic optimization techniques for global optimization running on a single-CPU. In addition, the more the number of CSs utilized in the PCS, the higher the search performance. For future research, the proposed PCS will be applied to various real-world and practical optimization problems including the scheduling problem (SP), assembly line balancing problem (ALBP), multiple traveling salesman problem (MTSP) and multiple vehicle routing problems (MVRP).

Acknowledgment. This paper was funded by King Mongkut's University of Technology North Bangkok with contract no. KMUTNB-64-DRIVE-25.

REFERENCES

[1] E. G. Talbi, *Metaheuristics: From Design to Implementation*, John Wiley & Sons, 2009.
[2] F. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2003.
[3] T. Ganesan, P. Vasant and I. Elamvazuthi, *Advances in Metaheuristics: Applications in Engineering Systems*, CSC Press Taylor & Francis Group, 2017.
[4] K. Lurang and D. Puangdownreong, Two-degree-of-freedom PIDA controllers design optimization for liquid-level system by using modified bat algorithm, *International Journal of Innovative Computing, Information and Control*, vol.16, no.2, pp.715-732, 2020.
[5] S. Sumpunsri and D. Puangdownreong, Multiobjective Lévy-flight firefly algorithm for optimal PIDA controller design, *International Journal of Innovative Computing, Information and Control*, vol.16, no.1, pp.173-187, 2020.
[6] N. Pringsakul and D. Puangdownreong, MoFPA-based PIDA controller design optimization for electric furnace temperature control system, *International Journal of Innovative Computing, Information and Control*, vol.16, no.6, pp.1863-1876, 2020.
[7] Y. Wang and X. Che, An improved water evaporation optimization algorithm, *International Journal of Innovative Computing, Information and Control*, vol.16, no.1, pp.107-122, 2020.
[8] S. El-Kenawy and M. Eid, Hybrid gray wolf and particle swarm optimization for feature selection, *International Journal of Innovative Computing, Information and Control*, vol.16, no.3, pp.831-844, 2020.
[9] E. Joelianto, A. Nainggolan and Y. A. Hidayat, Stingless bee algorithm for numerical optimization problems, *International Journal of Innovative Computing, Information and Control*, vol.16, no.6, pp.2063-2081, 2020.

[10] G. Seannery, Yacob, N. Chandra and D. David, Optimization of hospital patient management in hospitals with android-based applications, *ICIC Express Letters*, vol.14, no.3, pp.211-217, 2020.
[11] X. S. Yang and S. Deb, Cuckoo search via Lévy flights, *Proc. of the World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)*, pp.210-214, 2009.
[12] X. S. Yang, Cuckoo search and firefly algorithm: Overview and analysis, in *Cuckoo Search and Firefly Algorithm. Studies in Computational Intelligence*, Cham, Springer, 2014.
[13] X. S. Yang and S. Deb, Engineering optimisation by cuckoo search, *International Journal of Mathematical Modelling and Numerical Optimisation*, vol.1, no.4, pp.330-343, 2010.
[14] X. S. Yang and S. Deb, Multiobjective cuckoo search for design optimization, *Computers and Operations Research*, vol.40, no.6, pp.1616-1624, 2013.
[15] A. S. Joshi, O. Kulkarni, G. M. Kakandikar and V. M. Nandedkar, Cuckoo search optimization – A review, *Proc. of Materials Today*, vol.4, pp.7262-7269, 2017.
[16] I. Fister Jr., X. S. Yang, D. Fister and I. Fister, Cuckoo search: A brief literature review, in *Cuckoo Search and Firefly Algorithm. Studies in Computational Intelligence*, Cham, Springer, 2014.
[17] A. Gherboudj, A. Layeb and S. Chikhi, Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm, *International Journal of Bio-Inspired Computation*, vol.4, no.4, pp.229-236, 2012.
[18] A. Ouaarab, B. Ahiod and X. S. Yang, Discrete cuckoo search algorithm for the travelling salesman problem, *Neural Computing and Applications*, vol.24, nos.7-8, pp.1659-1669, 2014.
[19] K. Khan and A. Sahai, Neural-based cuckoo search of employee health and safety, *International Journal of Intelligent Systems and Applications*, vol.5, no.2, pp.76-83, 2013.
[20] A. Layeb, A novel quantum inspired cuckoo search for knapsack problems, *International Journal of Bio-Inspired Computation*, vol.3, no.5, pp.297-305, 2011.
[21] J.-H. Lin and I-H. Lee, Emotional chaotic cuckoo search for the reconstruction of chaotic dynamics, *Latest Advances in Systems Science & Computational Intelligence*, pp.123-128, 2012.
[22] M. Tuba, M. Subotic and N. Stanarevic, Modified cuckoo search algorithm for unconstrained optimization problems, *Proc. of the 5th European Conference on European Computing Conference*, pp.263-268, 2011.
[23] A. Ghodrati and S. Lotfi, A hybrid CS/GA algorithm for global optimization, *Proc. of the International Conference on Soft Computing for Problem Solving (SocProS-2011)*, pp.397-404, 2012.
[24] A. Ghodrati and S. Lotfi, A hybrid CS/PSO algorithm for global optimization, *Proc. of the Intelligent Information and Database Systems*, pp.89-98, 2012.
[25] M. Subotic, M. Tuba, N. Bacanin and D. Simian, Parallelized cuckoo search algorithm for unconstrained optimization, *Proc. of the 5th WSEAS Congress on Applied Computing Conference, and the 1st International Conference on Biologically Inspired Computation*, pp.151-156, 2012.
[26] T. Jitwang and D. Puangdownreong, Application of cuckoo search to robust PIDA controller design for liquid-level system, *International Journal of Innovative Computing, Information and Control*, vol.16, no.1, pp.189-205, 2020.
[27] M. M. Ali, C. Khompatraporn and Z. B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, *Journal of Global Optimization*, vol.31, pp.635-672, 2005.
[28] M. Jamil, X. S. Yang and H.-J. Zepernick, Test functions for global optimization: A comprehensive survey, *Swarm Intelligence and Bio-Inspired Computation Theory and Applications*, pp.193-222, 2013.