Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2014, Article ID 857254, 17 pages
http://dx.doi.org/10.1155/2014/857254

Research Article

An Effective Hybrid Cuckoo Search Algorithm with Improved Shuffled Frog Leaping Algorithm for 0-1 Knapsack Problems

Yanhong Feng,[1] Gai-Ge Wang,[2] Qingjiang Feng,[3] and Xiang-Jun Zhao[2]

[1] *School of Information Engineering, Shijiazhuang University of Economics, Shijiazhuang 050031, China*
[2] *School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, Jiangsu 221116, China*
[3] *School of Mathematical Science, Kaili University, Kaili, Guizhou 556011, China*

Correspondence should be addressed to Yanhong Feng; qinfyh@163.com

Received 4 June 2014; Revised 13 September 2014; Accepted 14 September 2014; Published 22 October 2014

Academic Editor: Saeid Sanei

An effective hybrid cuckoo search algorithm (CS) with improved shuffled frog-leaping algorithm (ISFLA) is put forward for solving 0-1 knapsack problem. First of all, with the framework of SFLA, an improved frog-leap operator is designed with the effect of the global optimal information on the frog leaping and information exchange between frog individuals combined with genetic mutation with a small probability. Subsequently, in order to improve the convergence speed and enhance the exploitation ability, a novel CS model is proposed with considering the specific advantages of Lévy flights and frog-leap operator. Furthermore, the greedy transform method is used to repair the infeasible solution and optimize the feasible solution. Finally, numerical simulations are carried out on six different types of 0-1 knapsack instances, and the comparative results have shown the effectiveness of the proposed algorithm and its ability to achieve good quality solutions, which outperforms the binary cuckoo search, the binary differential evolution, and the genetic algorithm.

1. Introduction

The application of nature-inspired metaheuristic algorithms to computational optimization is a growing trend [1]. Many hugely popular algorithms, including differential evolution (DE) [2, 3], harmony search (HS) [4, 5], krill herd algorithm (KH) [6–13], animal migration optimization (AMO) [14], grey wolf optimizer (GWO) [15], biogeography-based optimization (BBO) [16, 17], gravitational search algorithm (GSA) [18], and bat algorithm (BA) [19, 20], perform powerfully and efficiently in solving diverse optimization problems. Many metaheuristic algorithms have been applied to solve knapsack problems, such as evolutionary algorithms (EA) [21], HS [22], chemical reaction optimization (CRO) [23], cuckoo search (CS) [24–26], and shuffled frog-leaping algorithm (SFLA) [27]. To better understand swarm intelligence please refer to [28].

In 2003, Eusuff and Lansey firstly proposed a novel metaheuristic optimization method: SFLA, which mimics a group of frogs to search for the location that has the maximum amount of available food. Due to the distinguished benefit of its fast convergence speed, SFLA has been successfully applied to handle many complicated optimization problems, such as water resource distribution [29], function optimization [30], and resource-constrained project scheduling problem [31].

CS, a nature-inspired metaheuristic algorithm, is originally proposed by Yang and Deb in 2009 [32], which showed some promising efficiency for global optimization. Owing to the outstanding characteristics such as fewer parameters, easy implementation, and rapid convergence, it is becoming a new research hotspot in swarm intelligence. Gandomi et al. [33] first verified structural engineering optimization problems with CS algorithm. Walton et al. [34] proposed an improved cuckoo search algorithm which involved the addition of information exchange between the best solutions and tested their performance with a set of benchmark functions. Recently, the hybrid algorithms that combined CS with other

methods have been proposed and have become a hot topic studied by people, such as CS combined with a fuzzy system [35], a DE [36], wind driven optimization (WDO) [37], artificial neural network (ANN) [38], and genetic algorithm (GA) [39]. For details, see [40].

In 2011, Layeb [25] developed a variant of cuckoo search in combination with quantum-based approach to solve knapsack problems efficiently. Subsequently, Gherboudj et al. [24] utilized purely binary cuckoo search to tackle knapsack problems. A few scholars consider binary-coded CS and its performance need to further improve so as to further expand its fields of application. In addition, despite successful application to the solution of 0-1 knapsack problem by many methods, in fact, it is still a very active research area, because many existing algorithms do not cope well with some new and more intractable 0-1 knapsack problems hidden in the real world. Further, most of recently proposed algorithms focused on solving 0-1 knapsack problems with low dimension and medium dimension, but 0-1 knapsack problems with high dimension are involved little and the results are not highly satisfactory. What is more, the correlation between the weight and the value of the items may not be more concerned. This necessitates new techniques to be developed.

Therefore, in this work, we propose a hybrid CS algorithm with improved SFLA (CSISFLA) for solving 0-1 knapsack problem. To verify effectiveness of our proposed method, a large number of experiments on 0-1 knapsack problem are conducted and the experimental results show that the proposed hybrid metaheuristic method can reach the required optima more effectively than CS, DE, and GA even in some cases when the problem to be solved is too complicated and complex.

The rest of the paper is organized as follows. Section 2 introduces the preliminary knowledge of CS, SFLA algorithm, and the mathematical model of 0-1 KP problem. Then, our proposed CSISFLA for 0-1 KP problems is presented in Section 3. A series of simulation experiments are conducted in Section 4. Some conclusions and comments are made for further research in Section 5.

2. Review of the Related Work

In this section, the model of 0-1 knapsack problem and the basic CS and SFLA are introduced briefly.

2.1. 0-1 Knapsack Problem. The 0-1 knapsack problem, denoted by KP, is a classical optimization problem and it has high theoretical and practical value. Many practical applications can be formulated as a KP, such as cutting stock problems, portfolio optimization, scheduling problems, and cryptography. This problem has been proven to be a NP-hard problem; hence, it cannot be solved in a polynomial time unless $P = NP$ [44].

The 0-1 knapsack problem can be stated as follows:

$$\begin{aligned} \text{Maximize} \quad & f(x) = \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \le c, \\ & x_j = 0 \text{ or } 1, \quad j = 1, \ldots, n, \end{aligned} \tag{1}$$

where n is the number of items; w_j and p_j represent the weight and profit of item j, respectively. The objective is to select some items so that the total weight does not exceed a given capacity c, while the total profit is maximized. The binary decision variable x_i, with $x_i = 1$ if item i is selected, and $x_i = 0$ otherwise is used.

2.2. Cuckoo Search. CS is a relatively new metaheuristic algorithm for solving global optimization problems, which is based on the obligate brood parasitic behavior of some cuckoo species. In addition, this algorithm is enhanced by the so-called Lévy flights rather than by simple isotropic random walks.

For simplicity, Yang and Deb used the following three approximate rules [32, 45]:

(1) each cuckoo lays only one egg at a time and dumps its egg in a randomly chosen nest;

(2) the best nests with high-quality eggs will be carried over to the next generations;

(3) the number of available host nests is fixed, and the egg laid 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 simply abandon the nest and build a completely new nest.

The last assumption can be approximated by a fraction p_a of the n host nests which are replaced by new nests (with new random solutions).

New solution $\mathbf{X}_i^{(t+1)}$ is generated as (2) by using a Lévy flight [32]. Lévy flights essentially provide a random walk while their random steps followed a Lévy distribution for large steps which has an infinite variance with an infinite mean. Here the steps essentially form a random walk process with a power-law step-length distribution with a heavy tail as (3):

$$\mathbf{X}_i^{(t+1)} = \mathbf{X}_i^{(t)} + \alpha \oplus \text{Levy}(\lambda), \tag{2}$$

$$\text{Levy}(\lambda) \sim u = t^{-\lambda}, \tag{3}$$

where $\alpha > 0$ is the step size scaling factor. Generally, we take $\alpha = O(1)$. The product $\oplus$ means entry-wise multiplications.

2.3. Shuffled Frog-Leaping Algorithm. The SFLA is a metaheuristic optimization method that imitates the memetic evolution of a group of frogs while casting about for the location that has the maximum amount of available food [46]. SFLA, originally developed by Eusuff and Lansey in 2003, can be applied to handle many complicated optimization problems. In virtue of the beneficial combination of the genetic-based memetic algorithm (MA) and the social behavior-based PSO

algorithm, the SFLA has the advantages of global information exchange and local fine search. In SFLA, all virtual frogs are assigned to disjoint subsets of the whole population called memeplex. The different memeplexes are regarded as different cultures of frogs and independently perform local search. The individual frogs in each memeplex have ideas that can be effected by the ideas of other frogs and evolve by means of memetic evolution. After a defined number of memetic evolution steps, ideas are transferred among memeplexes in a shuffling process. The local search and the shuffling processes continue until defined convergence criteria are satisfied [47].

In the SFLA, the initial population P is partitioned into M memeplexes, each containing N frogs ($P = M \times N$). In this process, the ith goes to the jth memeplex where $j = i \bmod M$ (memeplex numbered from 0). The procedure of evolution of individual frogs contains three frog leapings. The position update is as follows.

Firstly, the new position of the frog individual is calculated by

$$Y = X + r_1 \times (B_k - W_k). \tag{4}$$

If the new position Y is better than the original position X, replace X with Y; else, another new position of this frog will perform in which the global optimal individual B_g replaces the best individual of kth memeplex B_k with the following leaping step size:

$$Y = X + r_2 \times (B_g - W_k). \tag{5}$$

If nonimprovement becomes possible in this case, the new frog is replaced by a randomly generated frog; else replace X with Y:

$$Y = L + r_3 \times (U - L). \tag{6}$$

Here, Y is an update of frog's position in one leap. r_1, r_2, and r_3 are random numbers uniformly distributed in $[0, 1]$. B_k and W_k are the best and the worst individual of the kth memeplex, respectively. B_g is the best individual in the whole population. U, L is the maximum and minimum allowed change of frog's position in one leap.

3. Hybrid CS with ISFLA for 0-1 Knapsack Problems

In this section, we will propose a hybrid metaheuristic algorithm integrating cuckoo search and improved shuffled frog-leaping algorithm (CSISFLA) for solving 0-1 knapsack problem. First, the hybrid encoding scheme and repair operator will be introduced. And then improved frog-leaping algorithm along with the framework of proposed CSISFLA will be presented.

3.1. Encoding Scheme. As far as we know, the standard CS algorithm can solve the optimization problems in continuous space. Additionally, the operation of the original CS algorithm is closed to the set of real number, but it does not have the closure property in the binary set $\{0, 1\}$. Based on above analysis, we utilize hybrid encoding scheme [26] and each cuckoo individual is represented by two tuples $\langle x_j, b_j \rangle$ ($j = 1, 2, \ldots, d$), where x_j works in the auxiliary search space and b_j performs in the solution space accordingly and d is the dimensionality of solution. Further, Sigmoid function is adopted to transform a real-coded vector $\mathbf{X}_i = (x_1, x_2, \ldots, x_d)^{\mathrm{T}} \in [-3.0, 3.0]^d$ to binary vector $\mathbf{B}_i = (b_1, b_2, \ldots, b_d)^{\mathrm{T}} \in \{0, 1\}^d$. The procedure works as follows:

$$b_i = \begin{cases} 1, & \text{if } \mathrm{Sig}(x_i) \geq 0.5, \\ 0, & \text{else}, \end{cases} \tag{7}$$

where $\mathrm{Sig}(x) = 1/(1 + e^{-x})$ is Sigmoid function.

The encoding scheme of the population is depicted in Table 1.

Table 1: Representation of population in CSISFLA.

$\langle \mathbf{X}_1, \mathbf{B}_1 \rangle$	$\langle \mathbf{X}_2, \mathbf{B}_2 \rangle$	$\cdots$	$\langle \mathbf{X}_i, \mathbf{B}_i \rangle$	$\cdots$	$\langle \mathbf{X}_n, \mathbf{B}_n \rangle$

3.2. Repair Operator. After evolving a generation, the feasibility of all the generated solutions is taken into consideration. That is, to say, the individuals could be illegal because of violating the constraint conditions. Therefore, a repair procedure is essential to construct illegal individuals. In this paper, an effective greedy transform method (GTM) is introduced to solve this problem [26, 48]. It cannot only effectively repair the infeasible solution but also can optimize the feasible solution.

This GTM consists of two phases. The first phase, called repairing phase (RP), checks each solution in order of decreasing p_i/w_i and confirms the variable value of one as long as feasibility is not violated. The second phase, called optimizing phase (OP), changes the remaining variable from zero to one until the feasibility is violated. The primary aim of the OP is to transform an abnormal chromosome coding into a normal chromosome, while the RP is to achieve the best chromosome coding.

3.3. Improved Shuffled Frog-Leaping Algorithm. In the evolution of SFLA, new individual is only affected by local optimal individual and the global optimal during the first two frog leapings, respectively. That is to say, there is a lack of information exchange between individuals and memeplexes. In addition, the use of the worst individual is not conducive to quickly obtain the better individuals and quick convergence. When the quality of the solution has not been improved after the first two frog leapings, the SFLA randomly generates a new individual without restriction to replace original individual, which will result in the loss of some valuable information of the superior individual to some extent. Therefore, in order to make up for the defect of the SFLA, an improved shuffled frog-leaping algorithm (ISFLA) is carefully designed and then embedded in the CSISFLA. Compared with SFLA, there are three main improvements.

The first slight improvement is that we get rid of sorting of the items according to the fitness value which will decrease in time cost.

The second improvement is that we adopt a new frog individual position update formula instead of the first two frog leapings. The idea is inspired by the DE/Best/1/Bin in DE algorithm. Similarly, each frog individual i is represented as a solution $\mathbf{X}_i$ and then the new solution Y is given by

$$Y = B_g \pm r_2 \times \left(B_k - X_{p1}\right), \tag{8}$$

where B_g is the current global best solution found so far. B_k is the best solution of the kth memeplex. X_{p1} is an individual of random selection with index of $p1 \neq i$ and r_2 is random number uniformly distributed in $[0, 1]$. In particular the plus or minus signs are selected with certain probability. The main purpose of improvement in (8) is to quicken convergence rate.

The third improvement is to randomly generate new individuals with certain probability instead of unconditional generating new individuals, which takes into consideration the retention of the better individuals in the population.

The main step of ISFLA is given in Algorithm 1. In Algorithm 1, P is the size of the population. M is the number of memeplex. D is the dimension of decision variables. And r_1 is a random real number uniformly distributed in (0, 1). And r_2, r_3, r_4, and p_m are all D-dimensional random vectors and each dimension is uniformly distributed in (0, 1). In particular, p_m is called probability of mutation which controls the probability of individual random initialization.

3.4. The Frame of CSISFLA. In this section, we will demonstrate how we combine the well-designed ISFLA with Lévy flights to form an effective CSISFLA. The proposed algorithm does not change the main search mechanism of CS and SFLA. In the iterative process of the whole population, Lévy flights are firstly performed and then frog-leaping operator is adopted in each memeplex. Therefore, the strong exploration abilities in global area of the original CS and the exploitation abilities in local region of ISFLA can be fully developed. The CSISFLA architecture is explained in Figure 1.

3.5. CSISFLA Algorithm for 0-1 Knapsack Problems. Through the design above carefully, the pseudocode of CSISFLA for 0-1 knapsack problems is described as follows (see Algorithm 2). It can be analyzed that there are essentially three main processes besides the initialization process. Firstly, Lévy flights are executed to get a cuckoo randomly or generate a solution. The random walk via Lévy flights is much more efficient in exploring the search space owing to its longer step length. In addition, some of the new solutions are generated by Lévy flights around the best solution, which can speed up the local search. Then ISFLA is performed in order to exploit the local area efficiently. Here, we regard the frog-leaping process as the process of cuckoo laying egg in a nest. The new nest generated with a probability p_m is far enough from the current best solution, which enables CSISFLA to avoid being trapped into local optimum. Finally, when an infeasible solution is generated, a repair procedure is adopted to keep feasibility and, moreover, optimize the feasible solution. Since the algorithm can well balance the exploitation and exploration, it expects to obtain solutions with satisfactory quality.

3.6. Algorithm Complexity. CSISFLA is composed of three stages: the sorting by value-to-weight ratio, the initialization, and the iterative search. The quick sorting has time complexity $O(P \log(P))$. The generation of the initial cuckoo nests has time complexity $O(P \times D)$. The iterative search consists of four steps (comment statements in Algorithm 2), and so forth, the Lévy flight, the first frog leaping, generate new individual and random selection which costs the same time $O(D)$. In summary, the overall complexity of the proposed CSISFLA is $O(P \log(P)) + O(P \times D) + O(D) = O(P \log(P)) + O(P \times D)$. It does not change compared with the original CS algorithm.

4. Simulation Experiments

4.1. Experimental Data Set. In existent researching files, cases studies and research of knapsack problems are about small-scale to moderate-scale problems. However, in real-world applications, problems are typically large-scale with thousands or even millions of design variables. In addition, the complexity of KP problem is greatly affected by the correlation between profits and weights [49–51]. However, few scholars pay close attention to the correlation between the weight and the value of the items. To test the validity of the algorithm for different types of instances, we adopt uncorrelated, weakly correlated, strongly correlated, multiple strongly correlated, profit ceiling, and circle data sets with different dimension. The problems are described as follows:

(i) uncorrelated instances: the weights w_j and the profits p_j are random integers uniformly distributed in $[10, 100]$;

(ii) weakly correlated instances: the weights w_j are random integers uniformly distributed in $[10, 100]$, and the profits p_j are random integer uniformly distributed in $[w_j - 10, w_j + 10]$;

(iii) strongly correlated instances: the weights w_j are random integers uniformly distributed in $[10, 100]$ and the profits p_j are set to $w_j + 10$;

(iv) multiple strongly correlated instances: the weights w_j are randomly distributed in $[10, 100]$. If the weight w_j is divisible by 6, then we set the $p_j = w_j + 30$ otherwise set it to $p_j = w_j + 20$;

(v) profit ceiling instances: the weights w_j are randomly distributed in $[10, 100]$ and the profits p_j are set to $p_j = 3\lceil w_j/3 \rceil$;

(vi) circle instances: the weights w_j are randomly distributed in $[10, 100]$ and the profits p_j are set to $p_j = d\sqrt{4R^2 - (w_j - 2R)^2}$. Choosing $d = 2/3$, $R = 10$.

For each data set, we set the value of the capacity. Consider $c = 0.75 \sum_{j=1}^{n} w_j$.

Begin
  For $i = 1$ *to* P ***do***
    $k = i \bmod M$
    select uniform randomly $p_1 \neq i$
    For $j = 1$ *to* D ***do***
      If $r_1 \geq 0.5$ ***then***
        $Y = B_g(j) + r_2 \times \left(B_k(j) - X_{p1}(j)\right)$
      Else
        $Y = B_g(j) - r_2 \times \left(B_k(j) - X_{p1}(j)\right)$
      End if
    End for
  If $f(Y) > f(X_i)$ ***then***
    $X_i = Y$
  Else If $r_3 \leq p_m$ ***then***
    $X_i = L + r_4 \times (U - L)$
    End if
  End if
End for
End

Algorithm 1: Improved shuffled frog-leaping algorithm.

Begin
Step 1. Sorting. *According to value-to-weight ratio* $p_i/w_i (i = 1, 2, 3, \ldots, n)$ *in descending order, a queue* $\{s_1, s_2, \ldots, s_n\}$ *of length n is formed.*
Step 2. Initialization. *Set the generation counter* $G = 1$*; Set probability of mutation* $p_m = 0.15$*. Generate P cuckoo nests randomly* $\{\langle \mathbf{X_1}, \mathbf{Y_1} \rangle, \langle \mathbf{X_2}, \mathbf{Y_2} \rangle, \ldots, \langle \mathbf{X_p}, \mathbf{Y_p} \rangle\}$*. Divide the whole population into M memeplexes, and each memeplex contains N (i.e.P/M) cuckoos; Calculate the fitness for each individual,* $f(\mathbf{Y_i})$*,* $1 \leq i \leq P$*, determine the global optimal individual* $< \mathbf{X_g}^{\mathbf{best}}, \mathbf{Y_g}^{\mathbf{best}} >$ *and the best individual of each memeplex* $\langle \mathbf{X_k}^{\mathbf{best}}, \mathbf{Y_k}^{\mathbf{best}} \rangle$*,* $1 \leq k \leq M$*.*
Step 3. While *the stopping criterionis not satisfied* ***do***
  For $i = 1$ *to* P
    $k = i \bmod M$
    select uniform randomly $p_1 \neq i$
    For $j = 1$ *to* D
      $\mathbf{X_i(j)} = \mathbf{X_i(j)} + \alpha \oplus \mathbf{Levy}(\lambda)$ *//Levy flight*
      If $r_1 \geq 0.5$ ***then*** *// The first frog leaping*
        $Temp = B_g(j) + r_2 \times \left(B_k(j) - X_{p1}(j)\right)$
      Else
        $Temp = B_g(j) - r_2 \times \left(B_k(j) - X_{p1}(j)\right)$
      End if
    End for
    If $f(Y_{temp}) > f(Y_i)$ ***then*** *// Generate new individual*
      $X_i = Temp$
    Else If $r_3 \leq FS$ ***then*** *// Random selection*
      $X_i = L + r_4 \times (U - L)$
      End if
    End if where $r_1, r_2, r_3, r_4 \sim U(0, 1)$
Repair the illegal individuals and optimize the legal individuals by performing GTM method
  End for
Keep best solutions.
Rank the solutions in descending order and find the current best $(\mathbf{Y^{best}}, f(\mathbf{Y^{best}}))$*.*
$G = G + 1$
Step 4. *Shuffle all the memeplexes*
Step 5. End while
End.

Algorithm 2: The main procedure of CSISFLA algorithm.

Table 2: Knapsack problem instances.

Problem	Correlation	Dimension	Target weight	Total weight	Total values
KP_1	Uncorrelated	150	6471	8628	8111
KP_2	Uncorrelated	200	8328	11104	10865
KP_3	Uncorrelated	300	12383	16511	16630
KP_4	Uncorrelated	500	20363	27150	28705
KP_5	Uncorrelated	800	33367	44489	44005
KP_6	Uncorrelated	1000	41948	55930	54764
KP_7	Uncorrelated	1200	49485	65980	66816
KP_8	Weakly correlated	150	6403	8538	8504
KP_9	Weakly correlated	200	8358	11144	11051
KP_{10}	Weakly correlated	300	12554	16739	16778
KP_{11}	Weakly correlated	500	20758	27677	27821
KP_{12}	Weakly correlated	800	33367	44489	44491
KP_{13}	Weakly correlated	1000	41849	55799	55683
KP_{14}	Weakly correlated	1200	49808	66411	56811
KP_{15}	Strongly correlated	300	12247	16329	19329
KP_{16}	Strongly correlated	500	21305	28407	33406
KP_{17}	Strongly correlated	800	33367	44489	52489
KP_{18}	Strongly correlated	1000	40883	54511	64510
KP_{19}	Strongly correlated	1200	50430	67240	79240
KP_{20}	Multiple strongly correlated	300	12908	17211	23651
KP_{21}	Multiple strongly correlated	500	20259	27012	37903
KP_{22}	Multiple strongly correlated	800	32767	43689	61140
KP_{23}	Multiple strongly correlated	1000	42442	56589	77940
KP_{24}	Multiple strongly correlated	1200	50222	66963	92653
KP_{25}	Profit ceiling	300	12666	16888	17181
KP_{26}	Profit ceiling	500	19811	26415	26913
KP_{27}	Profit ceiling	800	32011	42681	43497
KP_{28}	Profit ceiling	1000	42253	56337	57381
KP_{29}	Profit ceiling	1200	50208	66944	68157
KP_{30}	Circle	300	12554	16739	26448
KP_{31}	Circle	500	20812	27749	43880
KP_{32}	Circle	800	32581	43441	69527
KP_{33}	Circle	1000	42107	56143	88220
KP_{34}	Circle	1200	49220	65627	104287

Table 3: The effect of M and N on the performance of the CSISFLA.

Instance	N	$M = 2$				M	$N = 10$			
		Best	Worst	Mean	STD		Best	Worst	Mean	STD
KP_9	10	8727	**8704**	8711	5.5	2	**8727**	8704	8711	5.5
	15	**8728**	8701	**8715**	**6.8**	3	8725	8701	8713	7.0
	20	**8730**	8702	**8718**	6.5	4	8726	**8708**	8717	**6.3**
KP_{10}	10	13152	13124	13140	8.7	2	13152	13124	13140	8.7
	15	**13168**	13120	13144	12.6	3	13167	**13131**	**13146**	**8.2**
	20	**13174**	13126	**13148**	13.3	4	**13168**	**13128**	**13148**	**9.4**
KP_{11}	10	21820	21737	21773	22.1	2	21820	21737	21773	22.1
	15	21827	**21756**	**21786**	**17.3**	3	**21840**	21735	21783	24.6
	20	21814	**21757**	21778	**15.4**	4	**21848**	21742	**21788**	23.5

Table 4: The effect of p_m on the performance of the CSISFLA.

Instance	0	0.05	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
KP_1													
Best	7474	**7475**	**7475**	**7475**	**7475**	7474	**7475**	7474	7474	7474	7473	7474	7459
Worst	7430	7469	7468	**7471**	**7471**	7463	7457	7451	7451	7446	7437	7427	7407
Mean	7461	7473	**7474**	**7474**	7473	7471	7470	7468	7468	7461	7455	7448	7436
STD	12.60	1.50	1.57	**0.93**	1.27	3.57	4.96	6.03	5.87	8.83	10.11	11.17	13.88
KP_2													
Best	**9865**	**9865**	**9865**	**9865**	9863	9864	9860	9859	9850	9847	9844	9843	9842
Worst	9821	**9847**	9845	9844	9839	9823	9830	9818	9804	9778	9775	9768	9757
Mean	9847	**9858**	9856	9857	9852	9848	9847	9841	9833	9830	9812	9810	9783
STD	11.96	5.75	6.12	**5.32**	6.84	10.60	7.99	11.89	12.35	16.86	21.92	21.12	20.24
KP_8													
Best	6676	6674	6673	6672	6671	6672	6672	6671	**6678**	6666	6666	6662	6654
Worst	6658	6662	6663	**6665**	6662	6663	6662	6657	6655	6650	6652	6645	6642
Mean	6668	**6671**	6669	6669	6668	6668	6668	6664	6664	6659	6658	6652	6647
STD	4.59	2.95	2.59	**2.04**	2.44	2.79	2.39	4.17	4.45	4.06	3.88	4.27	3.17
KP_9													
Best	8730	**8734**	**8734**	8728	8731	8720	8723	8716	8712	8710	8707	8701	8688
Worst	**8707**	8703	8705	8701	8700	8702	8695	8684	8682	8675	8677	8664	8655
Mean	8716	**8718**	**8718**	8715	8714	8711	8707	8702	8697	8693	8690	8682	8676
STD	6.23	8.79	6.66	6.85	7.45	**4.59**	7.20	7.97	7.50	9.75	7.27	10.06	7.76

Table 5: Parameter settings of GA, DE, CS, and CSISFLA on 0-1 knapsack problems.

Algorithm	Parameter	Value
GA [41]	*Population size*	100
	Crossover probability	0.6
	Mutation probability	0.001
DE [42, 43]	*Population size*	100
	Crossover probability	0.9
	Amplification factor	0.3
CS [24]	*Population size*	40
	p_a	0.25
CSISFLA	M	4
	N	10
	p_m	0.15

Figures 2, 3, 4, 5, 6, and 7 illustrate six types of instances of 200 items, respectively.

The KP instances in this study are described in Table 2.

4.2. The Selection on the Value of M and N. The CSISFLA has some control parameters that affect its performance. In our experiments, we investigate thoroughly the number of subgroups M and the number of individuals in each subgroup N. The below three test instances are used to study the effect of M and N on the performance of the proposed algorithm. Firstly, M is set to 2, and then three levels of 10, 15, and 20 are considered for N (accordingly, the size of population is 2×10, 2×15, and 2×20). Secondly, a fixed individual number of each subgroup is 10, and the value of M is 2, 3, and 4, respectively. Results are summarized in Table 3.

As expected, with the increase of the individual number in the population, it is an inevitable consequence that there are more opportunities to obtain the optimal solution. This issue can be indicated by bold data in Table 3. In order to get a reasonable quality under the condition of inexpensive computational costs, we use $N = 10$ and $M = 4$ in the rest experiments.

4.3. The Selection on the Value of p_m. In this subsection, the effect of p_m on the performance of the CSISFLA is carefully investigated. We select two uncorrelated instances (KP_1, KP_2) and two weakly correlated instances (KP_8, KP_9) as the test instances for parameter setting experiment of p_m. For each instance, every test is run 30 times. We use $N = 10$, $M = 4$, and the maximum time of iterations is set to 5 seconds. Table 4 gives the optimization results of the CSISFLA using different values for p_m.

From the results of Table 4, it is not difficult to observe that the probability of mutation with $0.05 \leq p_m \leq 0.4$ is more suitable for all test instances which can be seen from data in bold in Table 3. In addition, the optimal solution dwindles steadily with the change of p_m from 0.5 to 1.0 and the worst results of four evaluation criteria are obtained when $p_m = 1$. Similarly, the performance of the CSISFLA is also poor when p_m is 0. As we have expected, 0 means that the position update in memeplex is completed entirely by the first Leapfrog, which cannot effectively ensure the diversity of the entire population, leading to the CSISFLA more easily fall into the local optimum, and 1 means that new individuals randomly generated without any restrictions which results in

Table 6: Experimental results of four algorithms with uncorrelated KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_1	GA	7316	6978	7200	7208	75.78
	DE	**7475**	7433	7471	7473	7.68
	CS	7472	7358	7403	7405	27.82
	CSISFLA	**7475**	**7467**	**7473**	**7474**	**1.56**
KP_2	GA	9673	9227	9503	9507	97.39
	DE	**9865**	9751	9854	**9865**	22.52
	CS	9848	9678	9737	9734	33.22
	CSISFLA	**9865**	**9837**	**9856**	9858	**7.23**
KP_3	GA	15022	14275	14756	14795	158.91
	DE	**15334**	15088	15287	15301	54.45
	CS	15224	15024	15092	15081	51.37
	CSISFLA	15327	**15248**	**15297**	**15302**	**18.48**
KP_4	GA	25882	25212	25498	25493	150.68
	DE	26333	25751	26099	26096	135.88
	CS	26208	25786	25936	25911	103.4
	CSISFLA	**26360**	**26193**	**26284**	**26277**	**38.54**
KP_5	GA	39528	38462	38976	39014	243.62
	DE	39652	39215	39410	39399	113.28
	CS	40223	39416	39565	39514	179.98
	CSISFLA	**40290**	**39885**	**40072**	**40081**	**91.97**
KP_6	GA	49072	47835	48483	48570	316.62
	DE	49246	48835	48989	48979	101.11
	CS	49767	49024	49164	49142	143.08
	CSISFLA	**49893**	**49567**	**49744**	**49737**	**97.52**
KP_7	GA	59793	58351	59135	59225	370.86
	DE	59932	59488	59707	59727	**110.39**
	CS	60629	59708	59939	59884	166.43
	CSISFLA	**60779**	**60264**	**60443**	**60420**	130.56

slow convergence. Generally speaking, using a small value of p_m is beneficial to strengthen the convergence ability and stability of the CSISFLA. The performance of the algorithm is the best when $p_m = 0.15$, so we will set $p_m = 0.15$ for the following experiments.

4.4. Experimental Setup and Parameters Setting. In this paper, in order to test the optimization ability of CSISFLA and further investigate effectiveness of the algorithms for different types of instance, we adopt a set of 34 knapsack problems (KP_1–KP_{34}). We compared the performance of CSISFLA with (a) GA, (b) DE, and (c) classical CS. In the experiments, the parameters setting are shown in Table 5.

In order to make a fair comparison, all computational experiments are conducted with Visual C++ 6.0. The test environment is set up on a PC with AMD Athlon(tm) II X2 250 Processor 3.01 GHz, 1.75 G RAM, running on Windows XP. The experiment on each instance was repeated 30 times independently. Further, best solution, worst solution, mean, median, and standard deviation (STD) for all the solutions are given in related tables. In addition, the maximum runtime was set to 5 seconds for the instances with dimension less than 500, and it was set to 8 seconds for other instances.

4.5. The Experimental Results and Analysis. We do experiment on 7 uncorrelated instances, 7 weakly correlated instances, and 5 other types of instances, respectively. The numerical results are given in Tables 6–11. The best values are emphasized in boldface. In addition, comparisons of the best profits obtained from the CSISFLA with those obtained from GA, DE, and CS for six KP instances with 1200 items are shown in Figures 8, 9, 10, 11, 12, and 13. Specifically, the convergence curves of four algorithms on six KP instances with 1200 items are also drawn in Figures 14, 15, 16, 17, 18, and 19. Through our careful observation, it can be analyzed as follows.

(a) Table 6 shows that CSISFLA outperforms GA, DE, and CS on almost all the uncorrelated knapsack instances in terms of computation accuracy and robustness. In particular, the best solution found by CSISFLA is slightly inferior to that obtained by DE on KP_3. On closer inspection, "STD" is much smaller than that of the other algorithms except for KP_7, which indicates the good stability of the CSISFLA and superior approximation ability.

Table 7: Experimental results of four algorithms with weakly correlated KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_8	GA	6627	6531	6593	6593	20.63
	DE	**6676**	6657	**6674**	**6676**	4.80
	CS	6660	6637	6648	6646	6.79
	CSISFLA	6673	**6663**	6668	6668	**2.23**
KP_9	GA	8658	8501	8588	8590	33.38
	DE	**8743**	**8743**	**8743**	**8743**	**0.00**
	CS	8717	8644	8676	8671	18.23
	CSISFLA	8728	8701	8714	8714	6.87
KP_{10}	GA	13062	12939	12997	12991	30.64
	DE	**13202**	**13158**	**13186**	**13186**	**9.76**
	CS	13157	13069	13094	13087	21.91
	CSISFLA	13168	13120	13145	13145	11.90
KP_{11}	GA	21671	21470	21571	21576	48.85
	DE	**21951**	21745	**21858**	**21859**	37.61
	CS	21935	21670	21746	21722	76.53
	CSISFLA	21827	**21756**	21788	21787	**16.66**
KP_{12}	GA	34587	34314	34488	34499	63.23
	DE	34814	34578	34721	34718	64.50
	CS	**34987**	34621	34697	34654	100.38
	CSISFLA	34818	**34721**	**34760**	**34758**	**22.87**
KP_{13}	GA	43241	42938	43082	43073	75.51
	DE	43327	43162	43217	43211	43.64
	CS	**43737**	43216	43340	43264	166.53
	CSISFLA	43409	**43312**	**43367**	**43368**	**27.23**
KP_{14}	GA	51472	50414	51058	51135	265.56
	DE	51947	51444	51600	51569	108.83
	CS	**53333**	51601	51831	51788	299.35
	CSISFLA	52403	**52077**	**52267**	**52264**	**86.19**

(b) From Table 7, it can be seen that DE obtained the best, mean, and median results for the first four cases, and CS attained the best results for the last three cases. Although the optimal solutions obtained by the CSISFLA are worse than DE or CS, the CSISFLA obtained the worst, median, and STD results in KP_{12}–KP_{14}, which still can indicate that the CSISFLA has better stability. Above all, the well-known NFL theorem [52] has stated clearly that there is no heuristic algorithm best suited for solving all optimization problems. Unfortunately, although weakly correlated knapsack problems are closer to the real world situations [49], the CSISFLA does not appear clearly superior to the other two algorithms in solving such knapsack problems.

(c) Obviously, in point of search accuracy and convergence speed, it can be seen from Table 8 that CSISFLA outperforms GA, DE, and CS on all five strongly correlated knapsack problems. If anything, the STD values tell us that CSISFLA is only inferior to CS.

(d) Similar results were found from Tables 9, 10, and 11 and it can be inferred that CSISFLA can easily yield superior results compared with GA, DE, and CS. The series of experimental results confirm convincingly the superiority and effectiveness of CSISFLA.

(e) Figures 8–13 show a comparison of the best profits obtained by the four algorithms for six types of 1200 items.

(f) Figures 14–19 illustrate the average convergence curves of all the algorithms in 30 runs where we can observe that CS and CSISFLA usually show the almost same starting point. However, CSISFLA surpasses CS in point of the accuracy and convergence speed. CS performs the second best in hitting the optimum. DE shows premature phenomenon in the evolution and does not offer satisfactory performance along with the extending of the problem.

Based on previous analyses, we can draw a conclusion that the superiority of CSISFLA over GA, DE, and CS in solving six types of KP instances is quite indubitable. In general, CS is slightly inferior to CSISFLA, so the next best is CS. DE and GA perform the third-best and the fourth-best, respectively.

Table 8: Experimental results of four algorithms with strongly correlated KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_{15}	GA	14785	14692	14754	14762	25.93
	DE	14797	14781	14789	14787	4.90
	CS	14804	14791	14797	**14797**	**2.43**
	CSISFLA	**14807**	**14795**	**14798**	**14797**	3.46
KP_{16}	GA	25486	25402	25458	25465	21.61
	DE	25502	25481	25492	25493	4.21
	CS	25514	25502	25506	25505	**3.49**
	CSISFLA	**25515**	**25505**	**25510**	**25512**	3.94
KP_{17}	GA	40087	39975	40039	40041	28.33
	DE	40111	40068	40089	40088	8.66
	CS	40107	40096	40103	40105	**3.88**
	CSISFLA	**40117**	**40098**	**40111**	**40113**	5.12
KP_{18}	GA	49332	49225	49300	49309	27.26
	DE	49363	49333	49346	49345	7.50
	CS	49380	49350	49364	49363	**7.04**
	CSISFLA	**49393**	**49362**	**49373**	**49373**	7.90
KP_{19}	GA	60520	60418	60482	60489	26.62
	DE	60540	60501	60519	60519	8.55
	CS	60558	60530	60542	60540	6.77
	CSISFLA	**60562**	**60539**	**60549**	**60550**	**5.70**

Table 9: Experimental results of four algorithms with multiple strongly correlated KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_{20}	GA	18346	18172	18284	18288	38.39
	DE	18387	18335	18354	18348	15.25
	CS	18386	18355	18368	18368	**4.73**
	CSISFLA	**18388**	**18368**	**18381**	**18386**	8.03
KP_{21}	GA	29525	29387	29461	29462	31.97
	DE	29548	29488	29519	29520	14.10
	CS	29589	29527	29555	29549	13.94
	CSISFLA	**29609**	**29562**	**29581**	**29585**	**12.38**
KP_{22}	GA	47645	47494	47568	47575	39.72
	DE	47704	47620	47659	47657	20.68
	CS	47727	47673	47696	47695	15.09
	CSISFLA	**47757**	**47697**	**47732**	**47736**	**13.02**
KP_{23}	GA	60529	60312	60455	60463	47.39
	DE	60572	60508	60534	60530	**13.98**
	CS	60607	60540	60576	60574	16.96
	CSISFLA	**60650**	**60579**	**60615**	**60612**	15.75
KP_{24}	GA	72063	71725	71914	71917	64.42
	DE	72072	71973	72018	72018	19.38
	CS	72094	72031	72058	72057	**15.93**
	CSISFLA	**72151**	**72070**	**72112**	**72111**	21.20

5. Conclusions

In this paper, we proposed a novel hybrid cuckoo search algorithm with improved shuffled frog-leaping algorithm, called CSISFLA, for solving 0-1 knapsack problems. Compared with the basic CS algorithm, the improvement of CSISFLA has several advantages. First, we specially designed an improved frog-leap operator, which not only retains the effect of the global optimal information on the frog leaping but also strengthens information exchange between frog individuals. Additionally, new individuals randomly generated with mutation rate. Second, we presented a novel

Table 10: Experimental results of four algorithms with profit ceiling KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_{25}	GA	12957	12948	12955	12957	2.53
	DE	12957	12951	12953	12954	1.83
	CS	12957	12954	12957	12957	0.76
	CSISFLA	**12957**	**12957**	**12957**	**12957**	**0.00**
KP_{26}	GA	20295	20268	20285	20286	7.37
	DE	20301	20292	20294	20294	2.17
	CS	20304	20295	20299	20298	**1.86**
	CSISFLA	**20307**	**20298**	**20304**	**20304**	2.28
KP_{27}	GA	32796	32769	32785	32787	6.99
	DE	32802	32793	32797	32796	**2.63**
	CS	32811	32799	32803	32802	3.12
	CSISFLA	**32820**	**32808**	**32812**	**32811**	3.34
KP_{28}	GA	43248	43215	43234	43236	8.76
	DE	43257	43245	43249	43248	3.57
	CS	43269	43251	43257	43254	4.41
	CSISFLA	**43272**	**43260**	**43266**	**43266**	**2.88**
KP_{29}	GA	51378	51348	51364	51366	7.25
	DE	51384	51372	51378	51378	**3.04**
	CS	**51399**	51378	51385	51384	4.32
	CSISFLA	**51399**	**51390**	**51396**	**51396**	3.10

Table 11: Experimental results of four algorithms with circle KP instances.

Instance	Algorithm	Best	Worst	Mean	Median	STD
KP_{30}	GA	21194	20899	21086	21096	71.44
	DE	21333	21192	21264	21277	32.46
	CS	21333	21194	21261	21261	**18.57**
	CSISFLA	**21333**	**21263**	**21300**	**21295**	34.04
KP_{31}	GA	35262	34982	35112	35124	82.25
	DE	35343	35184	35247	35267	38.08
	CS	35345	35271	35297	35277	31.29
	CSISFLA	**35414**	**35342**	**35354**	**35345**	**23.23**
KP_{32}	GA	55976	55451	55746	55771	116.83
	DE	56063	55914	55964	55954	44.95
	CS	**56280**	55988	56057	56061	55.01
	CSISFLA	56273	**56130**	**56185**	**56201**	**38.65**
KP_{33}	GA	70739	70247	70487	70456	113.53
	DE	70806	70641	70696	70684	**38.21**
	CS	70915	70729	70789	70797	42.50
	CSISFLA	**71008**	**70867**	**70924**	**70939**	41.17
KP_{34}	GA	83969	83339	83723	83757	142.75
	DE	84040	83820	83912	83899	56.64
	CS	**84645**	83954	84055	84033	121.94
	CSISFLA	84244	**84099**	**84175**	**84181**	**38.36**

CS model which is in an excellent combination with the rapid exploration of the global search space by Lévy flight and the fine exploitation of the local region by frog-leap operator. Third, CSISFLA employs hybrid encoding scheme; that is, to say, it conducts active searches in continuous real space, while the consequences are used to constitute the new solution in the binary space. Fourth, CSISFLA uses an effective GTM to assure the feasibility of solutions. The computational results show that CSISFLA outperforms the GA, DE, and CS in solution quality. Further,

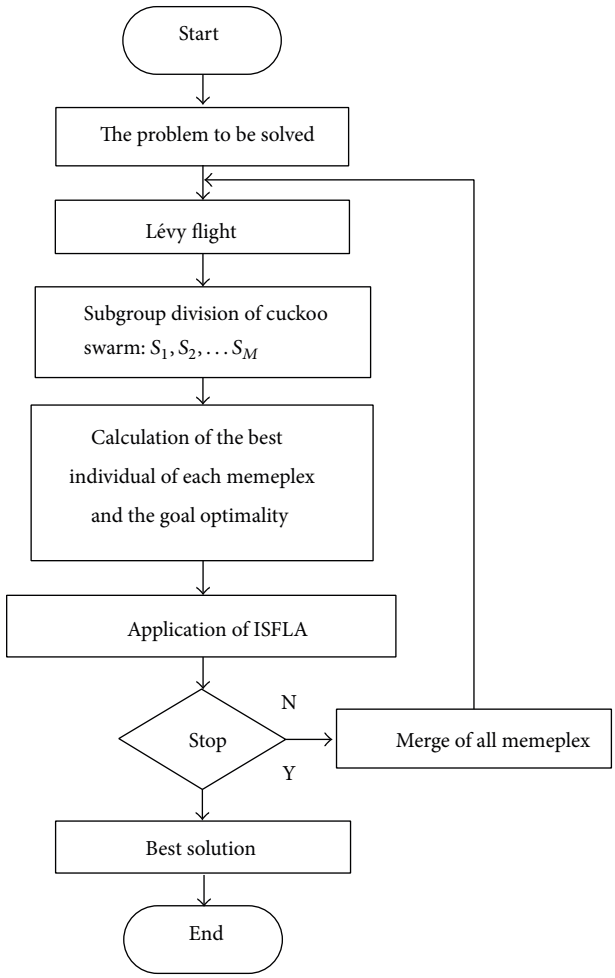

Figure 1: The architecture of CSISFLA algorithm.

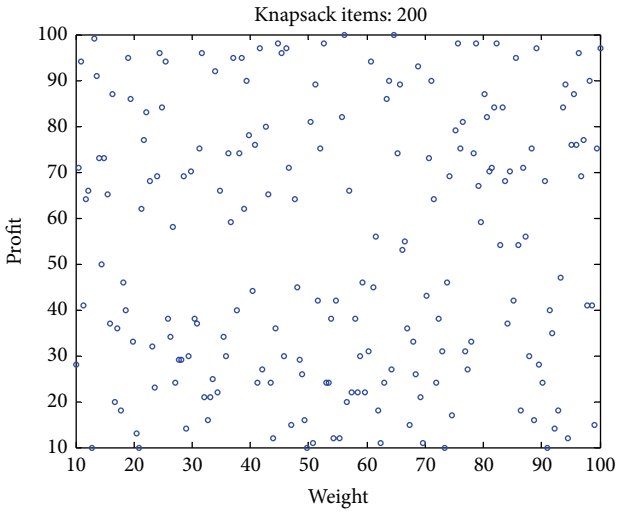

Figure 2: Uncorrelated items.

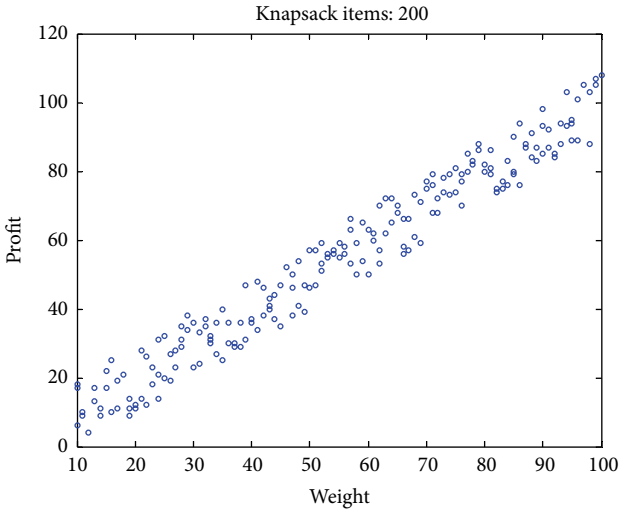

Figure 3: Weakly correlated items.

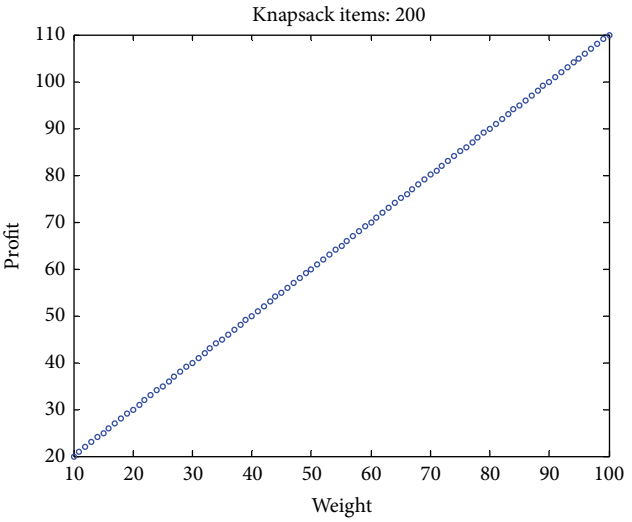

Figure 4: Strongly correlated items.

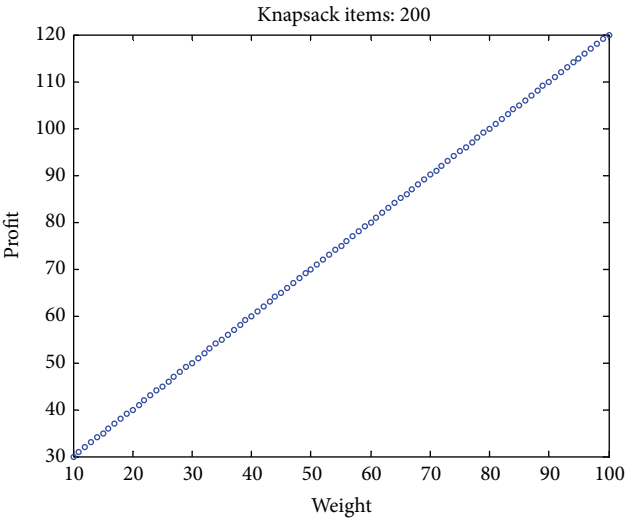

Figure 5: Multiple strongly correlated items.

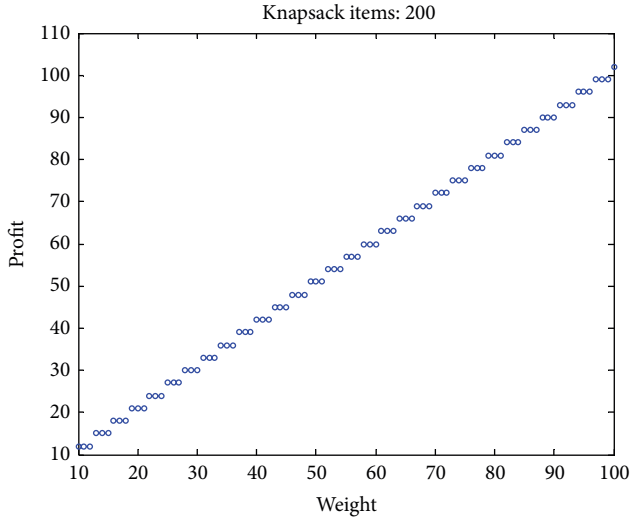

Figure 6: Profit ceiling items.

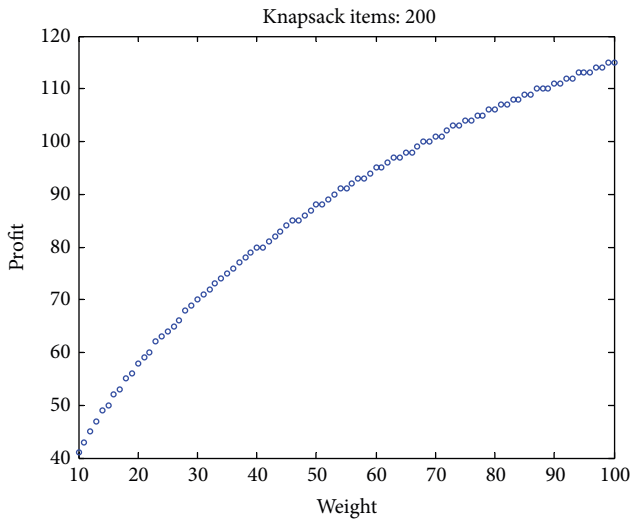

Figure 7: Circle items.

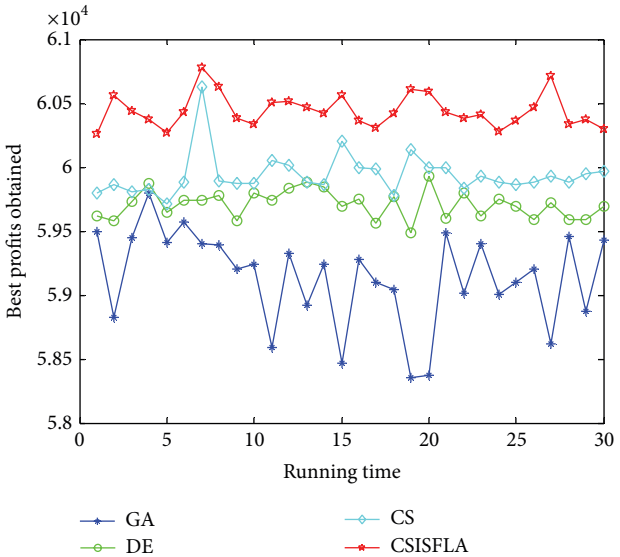

Figure 8: The best profits obtained in 30 runs for KP_7.

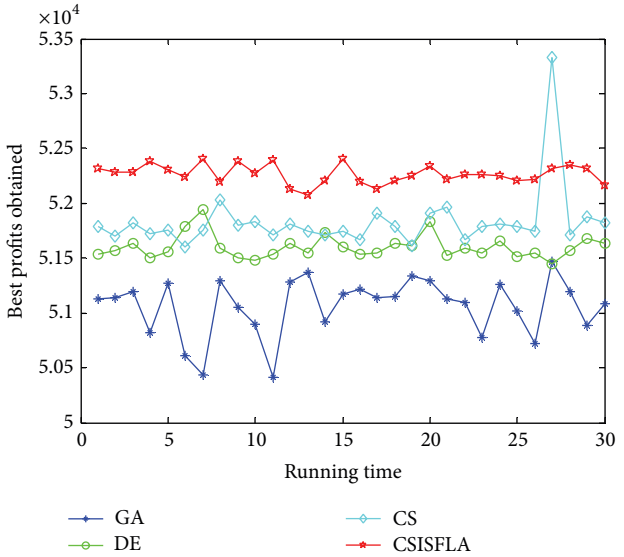

Figure 9: The best profits obtained in 30 runs for KP_{14}.

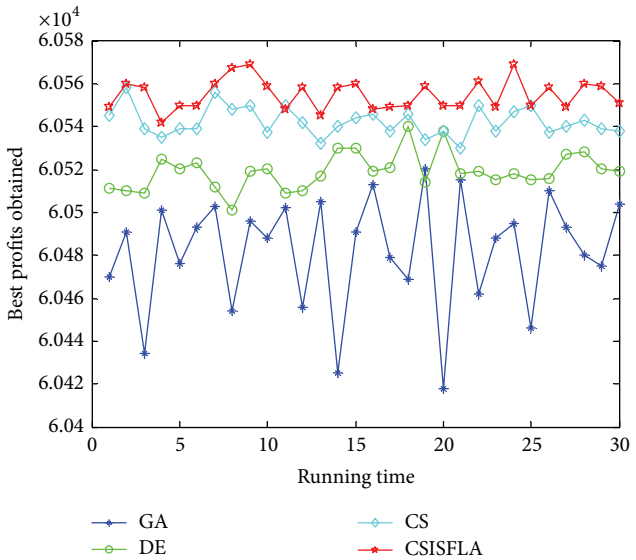

Figure 10: The best profits obtained in 30 runs for KP_{19}.

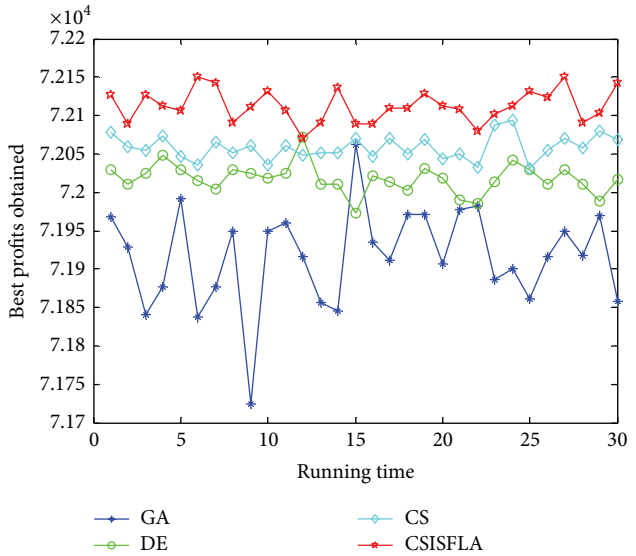

Figure 11: The best profits obtained in 30 runs for KP_{24}.

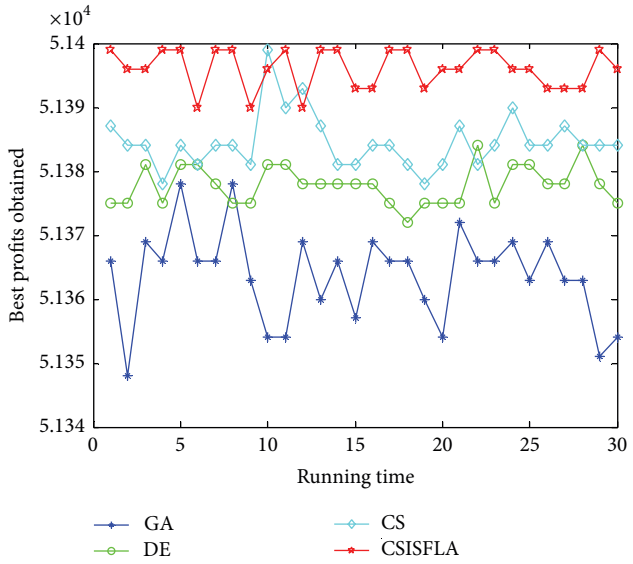

Figure 12: The best profits obtained in 30 runs for KP_{29}.

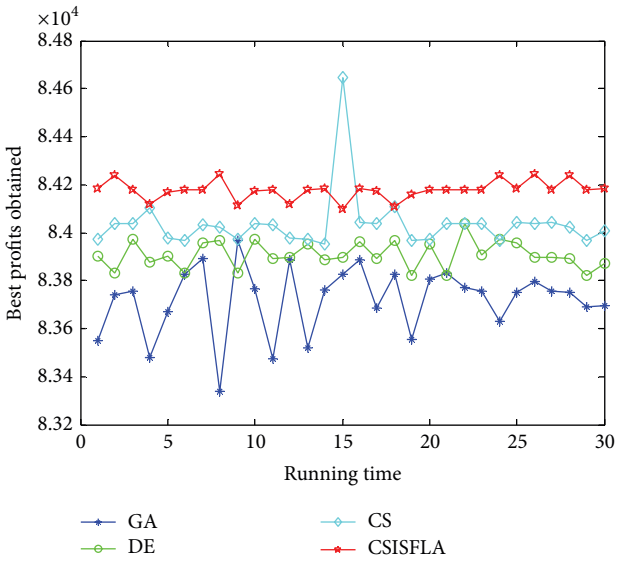

Figure 13: The best profits obtained in 30 runs for KP_{34}.

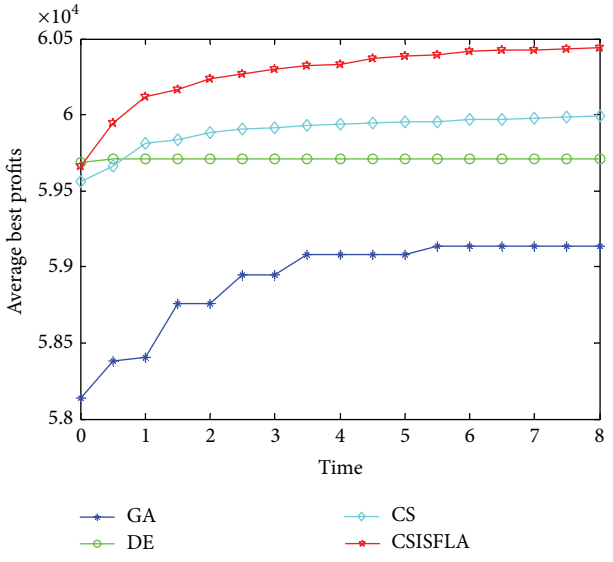

Figure 14: The convergence graphs of KP_{7}.

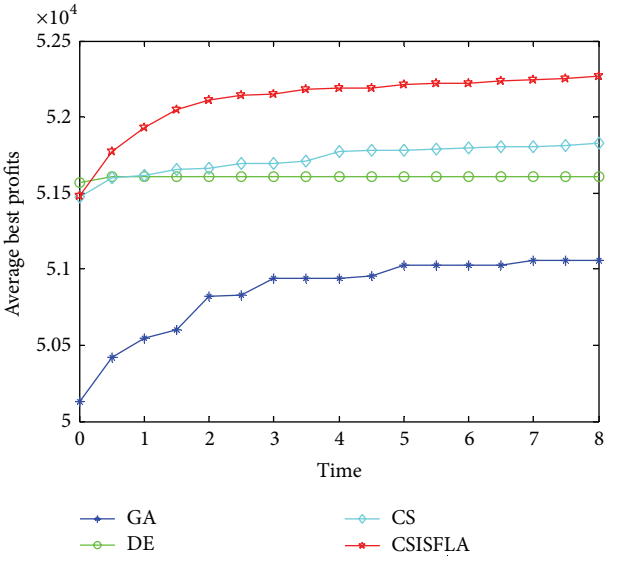

Figure 15: The convergence graphs of KP_{14}.

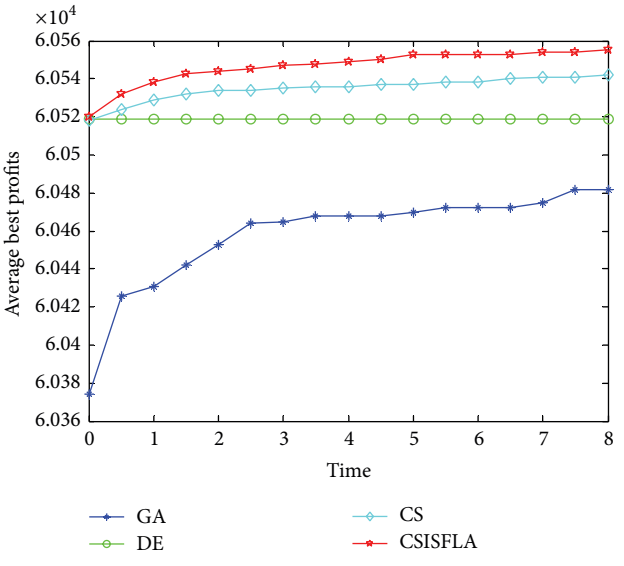

Figure 16: The convergence graphs of KP_{19}.

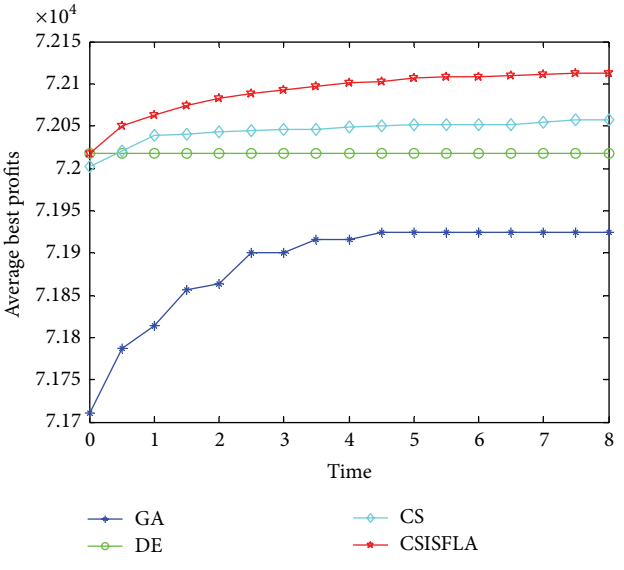

Figure 17: The convergence graphs of KP_{24}.

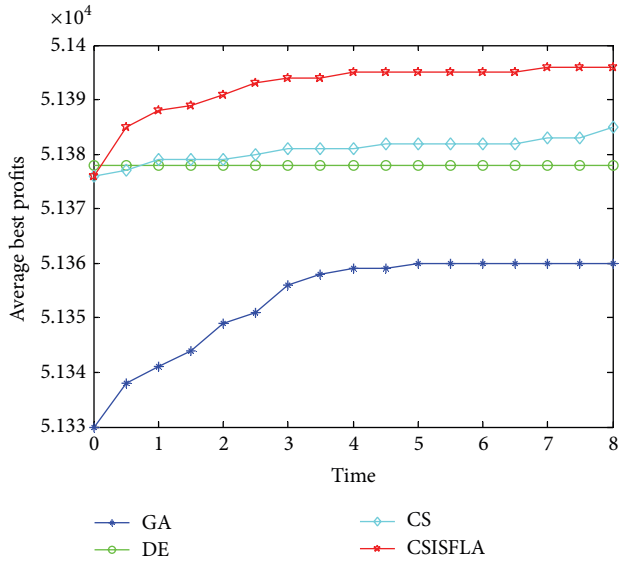

Figure 18: The convergence graphs of KP_{29}.

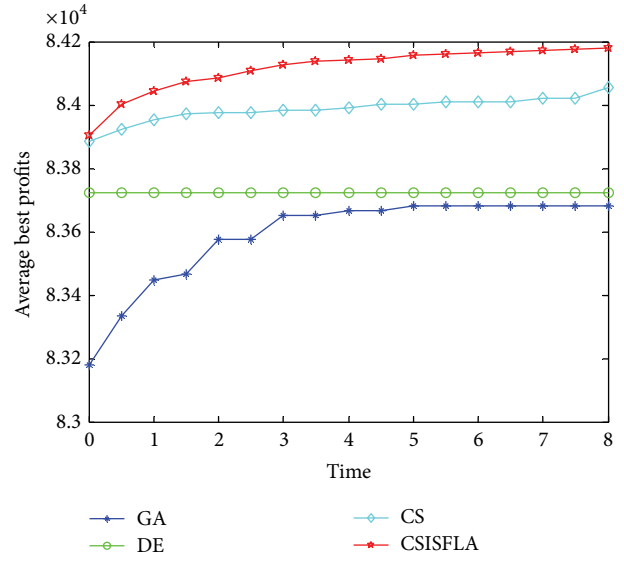

Figure 19: The convergence graphs of KP_{34}.

compared with ICS [26], the CSISFLA can be regarded as a combination of several algorithms and secondly the KP instances are more complex. The future work is to design more effective CS method for solving complex 0-1 KP and to apply the hybrid CS for solving other kinds of combinatorial optimization problems, multidimensional knapsack problem (MKP), and traveling salesman problem (TSP).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by Research Fund for the Doctoral Program of Jiangsu Normal University (no. 13XLR041) and National Natural Science Foundation of China (no. 61272297 and no. 61402207).

References

[1] X.-S. Yang, S. Koziel, and L. Leifsson, "Computational optimization, modelling and simulation: Recent trends and challenges," in *Proceedings of the 13th Annual International Conference on Computational Science (ICCS '13)*, vol. 18, pp. 855–860, June 2013.

[2] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[3] X. Li and M. Yin, "An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure," *Advances in Engineering Software*, vol. 55, pp. 10–31, 2013.

[4] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[5] L. Guo, G.-G. Wang, H. Wang, and D. Wang, "An effective hybrid firefly algorithm with harmony search for global numerical optimization," *The Scientific World Journal*, vol. 2013, Article ID 125625, 9 pages, 2013.

[6] A. H. Gandomi and A. H. Alavi, "Krill herd: a new bio-inspired optimization algorithm," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 12, pp. 4831–4845, 2012.

[7] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "An effective krill herd algorithm with migration operator in biogeography-based optimization," *Applied Mathematical Modelling*, vol. 38, no. 9-10, pp. 2454–2462, 2014.

[8] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "Stud krill herd algorithm," *Neurocomputing*, vol. 128, pp. 363–370, 2014.

[9] G. Wang, L. Guo, H. Wang, H. Duan, L. Liu, and J. Li, "Incorporating mutation scheme into krill herd algorithm for global numerical optimization," *Neural Computing and Applications*, vol. 24, no. 3-4, pp. 853–871, 2014.

[10] G.-G. Wang, L. Guo, A. H. Gandomi, G.-S. Hao, and H. Wang, "Chaotic krill herd algorithm," *Information Sciences*, vol. 274, pp. 17–34, 2014.

[11] G. G. Wang, A. H. Gandomi, A. H. Alavi, and G. S. Hao, "Hybrid krill herd algorithm with differential evolution for global numerical optimization," *Neural Computing and Applications*, vol. 25, no. 2, pp. 297–308, 2014.

[12] L. Guo, G.-G. Wang, A. H. Gandomi, A. H. Alavi, and H. Duan, "A new improved krill herd algorithm for global numerical optimization," *Neurocomputing*, vol. 138, pp. 392–402, 2014.

[13] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "A chaotic particle-swarm krill herd algorithm for global numerical optimization," *Kybernetes*, vol. 42, no. 6, pp. 962–978, 2013.

[14] X. Li, J. Zhang, and M. Yin, "Animal migration optimization: an optimization algorithm inspired by animal migration behavior," *Neural Computing and Applications*, vol. 24, no. 7-8, pp. 1867–1877, 2014.

[15] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.

[16] D. Simon, "Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702–713, 2008.

[17] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Let a biogeography-based optimizer train your multi-layer perceptron," *Information Sciences*, vol. 269, pp. 188–209, 2014.

[18] S. Mirjalili, S. Z. Mohd Hashim, and H. Moradian Sardroudi, "Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11125–11137, 2012.

[19] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pp. 65–74, Springer, Berlin, Germany, 2010.

[20] S. Mirjalili, S. M. Mirjalili, and X.-S. Yang, "Binary bat algorithm," *Neural Computing and Applications*, vol. 25, no. 3-4, pp. 663–681, 2013.

[21] R. Kumar and P. K. Singh, "Assessing solution quality of biobjective 0-1 knapsack problem using evolutionary and heuristic algorithms," *Applied Soft Computing Journal*, vol. 10, no. 3, pp. 711–718, 2010.

[22] D. Zou, L. Gao, S. Li, and J. Wu, "Solving 0-1 knapsack problem by a novel global harmony search algorithm," *Applied Soft Computing Journal*, vol. 11, no. 2, pp. 1556–1564, 2011.

[23] T. K. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem," *Applied Soft Computing Journal*, vol. 13, no. 4, pp. 1774–1780, 2013.

[24] 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.

[25] 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.

[26] Y. Feng, K. Jia, and Y. He, "An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems," *Computational Intelligence and Neuroscience*, vol. 2014, Article ID 970456, 9 pages, 2014.

[27] K. K. Bhattacharjee and S. P. Sarmah, "Shuffled frog leaping algorithm and its application to 0/1 knapsack problem," *Applied Soft Computing Journal*, vol. 19, pp. 252–263, 2014.

[28] R. S. Parpinelli and H. S. Lopes, "New inspirations in swarm intelligence: a survey," *International Journal of Bio-Inspired Computation*, vol. 3, no. 1, pp. 1–16, 2011.

[29] M. M. Eusuff and K. E. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225, 2003.

[30] X. Li, J. Luo, M.-R. Chen, and N. Wang, "An improved shuffled frog-leaping algorithm with extremal optimisation for continuous optimisation," *Information Sciences*, vol. 192, pp. 143–151, 2012.

[31] C. Fang and L. Wang, "An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem," *Computers and Operations Research*, vol. 39, no. 5, pp. 890–901, 2012.

[32] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proceedings of the World Congress on Nature and Biologically Inspired Computing (NABIC '09)*, pp. 210–214, December 2009.

[33] A. H. Gandomi, X.-S. Yang, and A. H. Alavi, "Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems," *Engineering with Computers*, vol. 29, no. 1, pp. 17–35, 2013.

[34] S. Walton, O. Hassan, K. Morgan, and M. R. Brown, "Modified cuckoo search: a new gradient free optimisation algorithm," *Chaos, Solitons and Fractals*, vol. 44, no. 9, pp. 710–718, 2011.

[35] K. Chandrasekaran and S. P. Simon, "Multi-objective scheduling problem: hybrid approach using fuzzy assisted cuckoo search algorithm," *Swarm and Evolutionary Computation*, vol. 5, pp. 1–16, 2012.

[36] G. G. Wang, L. H. Guo, H. Duan, H. Wang, L. Liu, and M. Shao, "A hybrid metaheuristic DE/CS algorithm for UCAV three-dimension path planning," *The Scientific World Journal*, vol. 2012, Article ID 583973, 11 pages, 2012.

[37] A. K. Bhandari, V. K. Singh, A. Kumar, and G. K. Singh, "Cuckoo search algorithm and wind driven optimization based study of satellite image segmentation for multilevel thresholding using Kapur's entropy," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3538–3560, 2014.

[38] M. Khajeh and E. Jahanbin, "Application of cuckoo optimization algorithm-artificial neural network method of zinc oxide nanoparticles-chitosan for extraction of uranium from water samples," *Chemometrics and Intelligent Laboratory Systems*, vol. 135, pp. 70–75, 2014.

[39] G. Kanagaraj, S. G. Ponnambalam, and N. Jawahar, "A hybrid cuckoo search and genetic algorithm for reliability-redundancy allocation problems," *Computers & Industrial Engineering*, vol. 66, no. 4, pp. 1115–1124, 2013.

[40] X. S. Yang and S. Deb, "Cuckoo search: recent advances and applications," *Neural Computing and Applications*, vol. 24, no. 1, pp. 169–174, 2014.

[41] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, Germany, 1996.

[42] S. Das and P. N. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[43] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied Soft Computing Journal*, vol. 11, no. 2, pp. 1679–1696, 2011.

[44] G. B. Dantzig, "Discrete-variable extremum problems," *Operations Research*, vol. 5, pp. 266–277, 1957.

[45] X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, Frome, UK, 2010.

[46] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.

[47] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, no. 1, pp. 43–53, 2005.

[48] Y. C. He, K. Q. Liu, and C. J. Zhang, "Greedy genetic algorithm for solving knapsack problems and its applications," *Computer Engineering and Design*, vol. 28, no. 11, pp. 2655–2657, 2007.

[49] S. Martello and P. Toth, *Knapsack Problems*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, NY, USA, 1990.

[50] D. Pisinger, *Algorithms for knapsack problems*, 1995.

[51] D. Pisinger, "Where are the hard knapsack problems?" *Computers & Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[52] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.