Appl Intell (2017) 46:158–179
DOI 10.1007/s10489-016-0822-y

Modified swarm intelligence based techniques for the knapsack problem

Kaushik Kumar Bhattacharjee[1] · **S. P. Sarmah**[1]

Published online: 3 August 2016

Abstract Swarm intelligence based algorithms have become an emerging field of research in recent times. Among them, two recently developed metaheuristics, cuckoo search algorithm (CSA) and firefly algorithm (FA) are found to be very efficient in solving different complex problems. CSA and FA are usually applied to solve the continuous optimisation problems. In this paper, an attempt has been made to utilise the merits of these algorithms to solve combinatorial problems, particularly 01 knapsack problem (KP) and multidimensional knapsack problem (MKP). In the improved version of CSA, a balanced combination of local random walk and the global explorative random walk is utilised along with the repair operator; whereas in the modified version of FA, the variable distance move with the repair operator of the local search and opposition-based learning mechanism is applied. Experiments are carried out with a large number of benchmark problem instances to validate our idea and demonstrate the efficiency of the proposed algorithms. Several statistical tests with recently developed algorithms from the literature present the superiority of these proposed algorithms.

Keywords Cuckoo search algorithm · Firefly algorithm · Particle swarm optimization · Knapsack problem · Performance metrics

S. P. Sarmah
spsarmah@iem.iitkgp.ernet.in

Kaushik Kumar Bhattacharjee
bhattacharjee.kaushik@gmail.com

1 Department of Industrial and Systems Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India

1 Introduction

Metaheuristic algorithms are developed to get near optimal solutions for difficult unsolvable complex problems. A large number of different types of metaheuristics are available in the literature. Metaheuristic inspired from nature, or natural phenomenon is called nature-inspired algorithm, and among them swarm intelligence (SI) based algorithms have received particular attention from the researchers [1, 22]. These algorithms are fast in nature, easy to understand and produce better solutions compared to the other classes of metaheuristics in most of the situations. SI techniques are based on the collective behaviour of swarms of bees, fish schools, and colonies of insects while searching for food, communicating with each other and socialising in their colonies. The SI models are based on self-organization, decentralisation, communication, and cooperation between the individuals within the team. The individual interaction is very simple but emerges as a complex global behaviour. It is the core of SI techniques [11]. In this paper, only the recently developed metaheuristics of SI class are considered, which are producing highly promising results for several optimisation problems. These are cuckoo search algorithm (CSA) and firefly algorithm (FA). Although SI based techniques have primarily been used and found very efficient in traditional optimisation problems, there is not much significant growth in the combinatorial optimisation field. Combinatorial problems are mostly NP-hard in nature, and we get very little information from their problem structure. So adapting these metaheuristic strategies in combinatorial solution space is a challenging job. This particular issue has motivated us to concentrate on this particular research problem. Here we present the solution process of single and multidimensional knapsack problem using the esteemed properties of CSA and FA and thereby contributed to the existing literature. Knapsack is one of the most intensively studied integer programming problems because of its simple structure. It allows exploitation of many combinatorial properties. Also, more complex optimisation problems can be solved through a series of knapsack-type subproblems. The most common form of zero-one knapsack problem (KP) is shown below,

$$\begin{aligned} &Maximize \quad f(x_1, x_2, ..., x_n) = \sum_{j=1}^{n} c_j x_j \\ &Subject\ to \quad \sum_{j=1}^{n} a_j x_j \le b, \\ &\qquad x_j \in \{0,1\}, \quad j = 1, 2, .., n \\ &\qquad c_j > 0, \quad a_j \ge 0, \quad b > 0. \end{aligned} \tag{1}$$

If there is more than one knapsack, then it is called multidimensional (multiple-constraint) knapsack problem (MKP), and its structure is given below.

$$\begin{aligned} &Maximize \quad f(x_1, x_2, ..., x_n) = \sum_{j=1}^{n} c_j x_j \\ &Subject\ to \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1, 2, ..., m \\ &\qquad x_j \in \{0,1\}, \quad j = 1, 2, .., n \\ &\qquad c_j > 0, \quad a_{ij} \ge 0, \quad b_i > 0. \end{aligned} \tag{2}$$

A set of n items with profits $c_j > 0$ and m resources with capacities $b_i > 0$ are given. Each item j consumes an amount $a_{ij} \ge 0$ from each resource i. The $0-1$ decision variables x_j indicate which items are selected. The objective function $f(x_1, x_2, ..., x_n)$ should be maximized subject to the constraints. Research has shown the wide applications of knapsack problems, such as capital budgeting problem, allocating processors and databases in a distributed computer system, project selection, cargo loading, and cutting stock problems. KP is an NP-complete problem though MKP is a strictly NP-hard combinatorial optimisation problem.

In the recent times, several metaheuristic algorithms are applied to solve 01 KP like genetic algorithm (GA) [67], schema-guiding evolutionary algorithm (SGEA) [39], quantum swarm evolutionary algorithm [59], ant colony optimization (ACO) [51], binary particle swarm optimization based on multi-mutation strategy (MMBPSO) [38], harmony search algorithm [69], etc. Similarly for solving 01 MKP instances, many heuristics and metaheuristics have been employed. Few examples are like tabu search (TS)[4], simulated annealing (SA) [20], GA [15], ACO [32], and particle swarm optimization (PSO) [54].

CSA combines the breeding behaviour of cuckoos with the Lévy's flight. It was first proposed by Yang and Deb in 2009 [64]. CSA is utilised to solve problems in diverse areas, and it includes mechanical engineering [66], automobile engineering [18], image processing [2]. CSA is also used for solving combinatorial problems like scheduling problem [13], traveling salesman problem (TSP) [44]. Authors have shown that CSA performs better than artificial bee colony in solving numerical optimisation problems [17]. Furthermore, CSA has been reported to converge faster than differential evolution [56]. Thus, CSA could be a potential metaheuristic algorithm for obtaining better performance in combinatorial optimisation problems.

FA is another recently developed SI method inspired by the flashing lights of fireflies in nature [61]. Like CSA, FA is also applied to solve many NP-hard problems, like TSP [30, 44], job shop scheduling problem (JSP) [35], quadratic assignment problem (QAP) [19]. In recent times, FA is utilised to solve different real-time applications like MRI brain segmentation [3], gray-scale image watermarking [43], optimal sensor placement in structural health monitoring [68], short-term load forecasting in electrical systems [31]. A comprehensive survey of FA can be found in [24].

Similar to CSA and FA, particle swarm optimisation is also an SI-based metaheuristic and developed by Kennedy and Eberhart [34]. It is inspired from the cooperation and communication of a swarm of birds. PSO finds applications in almost all types of optimisation problems. Also, several variants of PSO algorithms are applied to solve knapsack type problems, like binary particle swarm optimisation (BPSO) [33], genotype-phenotype modified binary particle swarm optimisation (GPMBPSO) [37], modified binary particle swarm optimisation (MBPSO) [5]. These variants are found to be very efficient in solving knapsack instances in the corresponding literature. So we choose PSO as the benchmark algorithm and also introduce a hybrid PSO algorithm to compare the experimental results.

As mentioned earlier, most of the metaheuristic strategies are developed concentrating on traditional optimisation problems in the continuous domain. Defining the neighbourhood solutions in combinatorial space and maintaining the key characteristics of metaheuristic, which govern the performance of the algorithm are the central theme of this study. For CSA, different transformation techniques are utilised to represent the solution structure. The combination of local random walk and the global explorative random walk is applied to define the neighbourhood solutions in combinatorial space. In FA, solutions are represented by the combinatorial structure, and the variable distance move along with the opposition-based learning mechanism is utilised to describe the neighbourhood solutions. The proposed algorithms are then applied to knapsack problem instances to validate the solution procedure and performance analysis. The work presented in this paper continues and expands our initial studies [9, 10]. More detailed descriptions of the algorithm structure along with the combination of local search techniques are added in this study. Solution procedure for MKP instances is also introduced here. Detailed study related to repair operator and parameter settings are discussed along with the statistical tests for validation.

The paper is organized as follows. First, a general framework of above-mentioned algorithms are discussed in Section 2, and then we focus on the implementation of these algorithms over knapsack problems in Sections 3 and 4. In Section 5, computational results and performance of these algorithms over knapsack problem are thoroughly investigated and discussed. Section 6 provides the concluding remarks and future direction of further study.

2 Background

The theoretical background required to develop the binary version of CSA and FA for solving knapsack problem is presented here.

2.1 Basic principle of cuckoo search algorithm

CSA is inspired by the reproduction strategy of cuckoos. Cuckoos lay their eggs in the nest of other host birds. If the host bird discovers that the eggs are not of its own, it will either throw these foreign eggs away or abandon its nest and build a new nest elsewhere. The general system equation of this algorithm is based on the generalised random walk pattern, given by

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \otimes L(s, \lambda), \tag{3}$$

where $x_i^{(t+1)}$ denotes $(t+1)$-th generation for i-th cuckoo; $x_i^{(t)}$ denotes the previous generation; $\alpha > 0$, represents step size which is related to the considered problem and $\otimes$ means entry-wise multiplications. The Lévy's flight essentially provides a random walk while the random step length is drawn from a Lévy distribution, given by

$$L(s, \lambda) = \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi} \times \frac{1}{s^{1+\lambda}}, \tag{4}$$

where Γ denotes gamma function. It has an infinite variance with an infinite mean. For the determination of the direction of the walk, the standard normal distribution is chosen. There is a probability that a particular egg may be discovered with a chance p_a. For this process,

$$X_i(new) = \begin{cases} X_i + \mathbf{r} \otimes (X_{r_1} - X_{r_2}), & \text{if } u \le p_a \\ X_i, & \text{otherwise.} \end{cases} \tag{5}$$

Here r_1 and r_2 are random positions; $\mathbf{r}$ and u are random numbers drawn from the uniform distribution. The basic procedure of CSA is shown in Algorithm 1.

Algorithm 1 Pseudocode for CSA procedure

```
initialize the basic parameters for CSA;
generate initial population of host nests;
evaluate the initial population;
rank the host nests;
while $g < MaxGen$ do
    for $i = 1 : n$ (all n host nests) do
        carry out levy flights to get new nests;
        evaluate the new nest;
        if new nest better than old nest then
            replace the old nest;
        end
    end
    a fraction of old population with worst nets are
    replaced by new nets;
    rank the nests and find the current best;
    update the global best if required;
    if termination condition fulfill then
        stop the evaluation process;
    end
end
return the global best solution;
```

2.2 Basic principle of firefly algorithm

In a typical FA, the brightness of a firefly means how well the objective of a solution is, and brighter firefly (solution) attracts other fireflies. As the distance between fireflies increases, the attractiveness decreases exponentially because of the light absorption by the medium. All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex [62].

i) **Distance between two fireflies**: The distance between any two fireflies (i and j) at x_i and x_j, respectively, can simply be evaluated by the Euclidean distance given by (6).

$$r_{ij} = ||x_j - x_j|| = \sqrt{\sum_{k=1}^{D}(x_{ik} - x_{jk})^2}, \tag{6}$$

where D is the dimension.

ii) **Attractiveness**: The attractiveness of a firefly at a distance r is given by the (7).

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \quad m \ge 1. \tag{7}$$

Here γ is the light absorption coefficient; β_0 is the attractiveness at distance $r = 0$, generally accepted as one; and m defines the sharpness of the exponential curve; usually one uses $m = 2$. For a given length scale Γ in an optimisation problem, Γ^{-m} is used as the initial value for γ.

iii) **Movement of firefly**: The movement of a firefly i attracted to another more attractive (brighter) firefly j is defined by (8).

$$x_i = x_i + \beta(r_{ij})(x_j - x_i) + \alpha(\epsilon - 0.5), \tag{8}$$

where ϵ is a random number drawn from the uniform distribution; and $\alpha \in [0, 1]$ is the scaling factor of the randomness.

Pseudocode for FA is given in Algorithm 2.

Algorithm 2 Pseudocode for FA procedure

```
initialize the basic parameters of FA;
generate the initial population of fireflies;
evaluate the initial population;
initialize the light intensity $l_i$;
while $g < MaxGen$ do
    for $i = 1 : n$ (all $n$ fireflies) do
        for $j = 1 : n$ (all $n$ fireflies) do
            if $l_j > l_i$ then
                move firefly $i$ towards firefly $j$;
            end
            attractiveness varies with the distance;
        end
        evaluate new solution;
        update the light intensity $l_i$;
    end
    rank the fireflies and find out the current best;
    update the global best if required;
    if termination criteria fulfill then
        stop the evaluation process;
    end
end
return the global best solution;
```

3 Binary cuckoo search algorithm

CSA was originally designed to handle a continuous problem. So one cannot directly apply CSA to knapsack problem. For this reason, we modified the original CSA procedure, and the modified binary cuckoo search algorithm (BCSA) is discussed in detail in this section.

3.1 Construction of individual nest

The knapsack problem is an integer programming problem. There are two possible values of decision variable x_j, zero or one. The individual nest (solution) is represented by an n-bit binary string, where n is the number of variables in knapsack problem. A bit string $x \in \{0, 1\}^n$, might represent an infeasible solution.

For 01 KP, the fitness function of individual nest is defined as

$$f(\mathbf{x}) = \mathbf{c} \times \mathbf{x} - \rho(\mathbf{a} \times \mathbf{x} - b)^2, \tag{9}$$

where $\rho = max_{j=1,2,..,n} \frac{c_j}{a_j}$ considering $a_j > 0$ in (1).

Similarly, for 01 MKP, we represent the fitness function as given in (10).

$$f(x) = \sum_{j=1}^{n} c_j x_j - s \times [max(\mathbf{c})]^2, \tag{10}$$

where s represents the number of violated constraints and $\mathbf{c}$ is the profit vector in (2).

Similar to other metaheuristics, the initial locations of the nests are scattered over the solution space randomly. We define the stopping rule based on four basic criteria: reach the optimal solution, maximum run time, maximum number of iterations, and predefined number of non-improvement iterations. If any of these criteria is satisfied, the algorithm stops.

3.2 Evaluating new nest

New solutions are generated using the Léavy's flight in original CSA. Here a balanced combination of a local random walk and the global explorative random walk controlled by switching parameter p_a is used as it performs better than simple Léavy's flight as suggested in [21, 65]. The local random walk is defined by

$$x_i^{(t+1)} = x_i^{(t)} + \alpha s \otimes H(p_a - \epsilon) \otimes \left(x_j^{(t)} - x_k^{(t)}\right), \tag{11}$$

where $x_j^{(t)}$ and $x_k^{(t)}$ are two different solutions selected by random permutation at generation t, $H(u)$ is a Heaviside function, $\epsilon \sim U(0, 1)$, s is the scaling factor, and $\alpha > 0$ is the step size.

The global random walk is same as given in the (3). Here generation of random numbers with Lévy's flight is done by using Mantegna algorithm [42] for a symmetric Lévy stable distribution [63]. The random variable is generated by,

$$w_j = (\frac{u_j}{v_j})^{\frac{1}{\beta}} \qquad \forall j, \tag{12}$$

where $\beta = 1.50$ and u_j and v_j are drawn from $N(0, \sigma^2)$ and $N(0, 1)$ respectively; and σ value is defined by,

$$\sigma = \left(\frac{\Gamma(1+\beta) \times sin(\pi\beta/2)}{\Gamma((\frac{1+\beta}{2}) \times \beta \times 2^{\frac{\beta-1}{2}})}\right)^{\frac{1}{\beta}}. \tag{13}$$

3.3 Binary variable handling

Here three different approaches are used for handling binary variables, and these are various transformation function type. The general equation for both the local and global walk is given by

$$x_i^{(t+1)} = x_i^{(t)} + \delta_i^{(t)}, \tag{14}$$

where $\delta_i^{(t)}$ is the step size for the local or global walk given in (11) and (13), respectively. For symmetric Lévy distribution $\delta_i^{(t)}$ can be positive or negative. At first we consider that $\delta_i^{(t)}$ is bounded, i.e. $-\delta_0 \le \delta_i^{(t)} \le \delta_0$, where δ_0 is maximum allowable change. As $x_i^{(t)}$ is either zero or one, so we define $x_i^{(t+1)}(new)$ as

$$x_i^{(t+1)}(new) = \begin{cases} 0 & \text{if} \quad x_i^{(t+1)} \le 0, \\ \left[x_i^{(t+1)} \right] & \text{if} \quad 0 < x_i^{(t+1)} < 1, \\ 1 & \text{otherwise.} \end{cases} \tag{15}$$

In the second case, we consider that

$$x_i^{(t+1)}(new) = \begin{cases} 1 \text{ if} \quad u < sigm\left(x_i^{(t+1)}\right), \\ 0 \text{ otherwise.} \end{cases} \tag{16}$$

In the last case, we consider

$$x_i^{(t+1)}(new) = \begin{cases} 1 \text{ if} \quad u < \frac{x_i^{(t+1)}+\delta_0}{1+2\delta_0}, \\ 0 \text{ otherwise.} \end{cases} \tag{17}$$

The term u is a uniform random number ($u \sim U(0,1)$), $sigm(z)$ is a sigmoid limiting transformation having "S" shape curve function, and defined as $sigm(z) = \frac{1}{1+exp(-\gamma z)}$; where γ controls the steepness of the sigmoid function.

3.4 Local search

After evaluating new nest, Stochastic Hill Climbing technique was performed to detect the best solution in the neighbourhood. In the maximisation case, it is done if only the new solution is better than the current one. This process is repeated until no further improvement is possible or when a predefined number of steps are reached. The pseudo code of Stochastic Hill Climbing is given in Algorithm 3. For generating a new solution from the current one, we use a random position in the nest and change its value. Stochastic Hill Climbing is a simple local search method to find out local maxima. This procedure generates solutions of high quality without introducing vast computational effort.

Algorithm 3 Pseudocode for stochastic hill climbing

```
current solution $s$ from new nest;
evaluate the objective function $f(s)$;
$stop = FALSE$;
$iter = 0$;
while $stop = FALSE$ do
    choose a random solution in the neighborhood $s_n$;
    if $f(s_n) > f(s)$ then
        replace $s$ by $s_n$;
    else
        $iter = iter + 1$;
    end
    if $iter = Iter_{max}$ then
        $stop = TRUE$;
    end
end
return $s$;
```

3.5 Constraint handling

When the binary string violates the constraint of the given problem, then repair algorithm is employed to make infeasible solutions to feasible one. This procedure depends on the problem type and the constraint.

For 01 KP, variables are sorted and renumbered according to the decreasing order of their profit to weight ratios. The greedy algorithm is utilised for selection procedure which always chooses the last item for deletion.

In the case of 01 MKP at the initialization step, variables are sorted and renumbered according to the decreasing order of their pseudo-utility, and it is calculated by the surrogate duality approach introduced by Pirkul [46]. The general idea of this approach is described briefly.

The surrogate relaxation problem of the 01 MKP can be defined as:

$$\begin{aligned} &Maximize \quad \sum_{j=1}^{n} c_j x_j \\ &Subject\ \ to \quad \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \omega_i a_{ij} \right) x_j \le \sum_{i=1}^{m} \omega_i b_i, \quad \forall i \\ &\qquad\qquad x_j \in \{0,1\}, \quad j = 1, ..., n, \quad i = 1, ..., m \end{aligned} \tag{18}$$

where $\omega = \{\omega_1, \omega_2, ..., \omega_m\}$ is a set of surrogate multipliers (or weights) of some positive real numbers. These weights are obtained by using a simple method suggested in [46], in which the LP relaxation of the original 01 MKP is solved, and the values of the dual variables are viewed as the weights. The weight ω_i can be seen as the shadow price of the i-th constraint in the LP relaxation of the 01 MKP. The pseudo-utility ratio for each variable, based on the surrogate constraint coefficient, is defined as

$$u_j = \frac{c_j}{\sum_{i=1}^{m} \omega_i a_{ij}}. \tag{19}$$

The repair operator is inspired by the idea of Chu and Beasley [15], which consists of two phases. The first phase (called DROP) examines each bit of the solution string in the increasing order of u_j and changes a bit from one to zero if feasibility is violated. The second phase (called ADD) reverses the process by examining each bit in decreasing order of u_j and changes a bit from zero to one as long as feasibility is not violated. The pseudo-code of the repair operator is given in Algorithm 4.

Algorithm 4 Pseudocode for a repair operator

```
let: $R_i$ = the accumulated resources of constraint $i$ in $x$;
initialize $R_i = \sum_{j=1}^{n} a_{ij}x_j, \forall i \in \{1,2,..,m\}$;
for $j = n$ to $1$ do
    /* DROP Phase                                          */
    if $(x_j = 1)$ and $(R_i > b_i$, for any $i \in \{1,2,..,m\})$ then
        $x_j := 0$;
        $R_i := R_i - a_{ij}$, $\forall i \in \{1,2,..,m\}$;
    end
end
for $j = 1$ to $n$ do
    /* ADD Phase                                           */
    if $(x_j = 0)$ and $(R_i + a_{ij} \le b_i$, $\forall i \in \{1,2,..,m\})$ then
        $x_j := 1$;
        $R_i := R_i + a_{ij}$, $\forall i \in \{1,2,..,m\}$;
    end
end
```

4 Binary firefly algorithm

Several studies are available in the literature of FA for continuous optimisation problems. But only a few studies are available for integer programming problem. In a recent work by Palit et al. [45], a binary firefly algorithm was used for cryptanalysis of knapsack cypher algorithm so as to deduce the meaning of an encrypted message. Binary adaptive firefly algorithm was used for fault identification in parallel and distributed system by Falcon et al. [23]. Chandrasekaran and Simon [12] used binary real coded firefly algorithm for solving network and reliability constrained unit commitment problem. In all these works they use the sigmoid function for transforming continuous variables to binary variables, and use Euclidean distance to perform light intensity measurement. Recently, Baykasoglu and Ozsoydan [6] solved the dynamic multidimensional knapsack problem using FA. They used priority based encoding technique for mapping continuous solutions to the combinatorial domain and replace the pairwise comparison with a specialised comparison using tuning parameter depending upon the iteration number. Here we use a different approach for solving knapsack problems similar to the approach used in [47] for solving facility location problem. It does not involve any encoding technique and utilizes hamming distance to measure the light intensity.

4.1 Construction of individual firefly

The individual firefly (solution) is represented by an n-bit binary string, where n is the number of variables in knapsack problem. A bit string $\mathbf{x} \in \{0, 1\}^n$, may represent an infeasible solution; so the fitness function of individual firefly for 01 KP is defined as earlier, given by (9). Similarly for 01 MKP, we represent the fitness function given by (10). The initial population is generated randomly from the entire dimension to achieve sufficient diversification. The stopping rule is defined based on four basic criteria: reach the optimal solution, maximum run time, maximum number of iterations, predefined number of non-improvement iterations. If any of these criteria is satisfied, the algorithm stops.

Table 1 Comparison between different types of transformation functions used for binary variable handling of MKP

f	Criteria					
	best	worst	avg	median	std	ATT
f_1	3800	3800	3800	3800	0	0.007
	3800	3800	3800	3800	0	0.003
	3800	3800	3800	3800	0	0.005
f_2	8706.1	8577.8	8639.6	8650.1	47.49	1.049
	8706.1	8706.1	8706.1	8706.1	0	0.024
	8706.1	8706.1	8706.1	8706.1	0	0.025
f_3	4015	4005	4010.2	4015	5.05	0.681
	4015	4015	4015	4015	0	0.016
	4015	4015	4015	4015	0	0.028
f_4	6120	6090	6115.8	6120	8.83	0.439
	6120	6120	6120	6120	0	0.023
	6120	6120	6120	6120	0	0.024
f_5	12400	12400	12400	12400	0	0.069
	12400	12400	12400	12400	0	0.031
	12400	12400	12400	12400	0	0.046
f_6	10584	10547	10550	10547	9.21	1.762
	10618	10584	10600	10603	12.46	2.407
	10618	10570	10592	10587	10.91	2.435
f_7	16501	16436	16496	16499	12.73	2.153
	16537	16499	16503	16499	8.06	2.683
	16524	16499	16502	16499	5.68	2.726

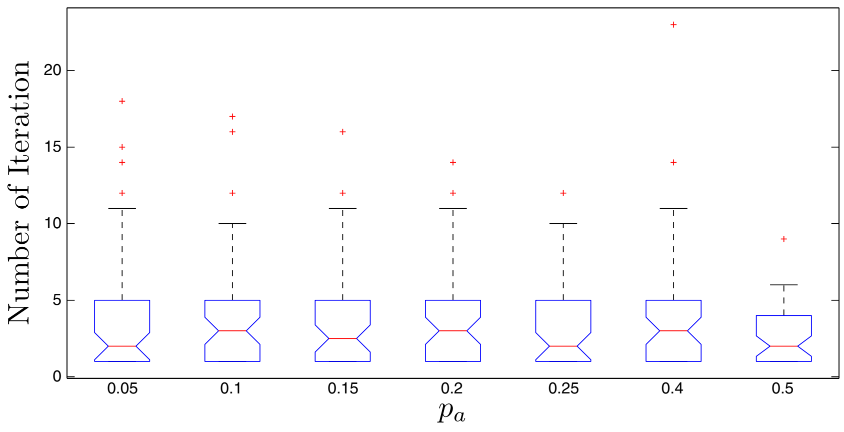

Fig. 1 The effect of p_a for MKP instance f_4

4.2 Distance of two fireflies

The distance between any two fireflies i and j is defined by the Hamming distance, i.e., the number of different elements between their permutations. The difference between the objective function values is directly proportional to the Hamming distance.

4.3 Attractiveness of fireflies

A firefly i is attracted to another firefly j if the objective function value of i is smaller than the objective function value of j. The attractiveness of firefly i on firefly j is given by

$$\beta(r_{ij}) = \beta_0 e^{-\gamma r_{ij}^2}, \tag{20}$$

where r_{ij} is the Hamming's distance between fireflies i and j. Theoretical value of the light absorption coefficient γ is $\gamma \in [0, \infty]$, we have considered γ in the range of $[0.01, 0.20]$.

4.4 Movement of fireflies

In the continuous problem, the movement is defined by the (8), which can be represented by (21a–21b), like [19] in the discrete case

$$x_i = (1 - \beta(r_{ij}))x_i + \beta(r_{ij})x_j, \tag{21a}$$
$$x_i = x_i + \alpha(\epsilon - 0.5). \tag{21b}$$

Any firefly moves in solution space in two steps: in the first step $\beta(r_{ij})$ determines the motion of firefly i towards firefly j given by (21a); then in the next step, α determines the random movement of firefly i provided by (21b). It is important to note that these two phases are not interchangeable.

The first step is known as β-step and from the equation, it is clear that x_i will be equal to x_j with a probability given by $\beta(r_{ij})$, and x_i will be unchanged with a probability given by $(1 - \beta(r_{ij}))$. The β-step procedure is shown in Algorithm 5.

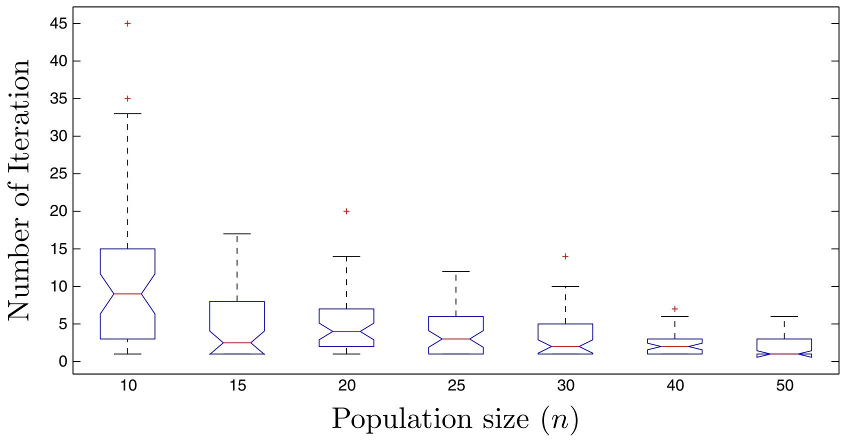

Fig. 2 The effect of population size n for MKP instance f_4

Table 2 Kruskal-Wallis ANOVA table

Source	SS	df	MS	χ^2	$p > \chi^2$
Groups	562237	6	93706.2	57.35	1.55E-10
Error	2859316.5	343	8336.2		
Total	3421553.5	349			

Algorithm 5 Pseudocode for β Step

```
evaluate the objective function values of firefly $x_i$ and $x_j$, $f(x_i)$ and $f(x_j)$ respectively;
if $f(x_i) < f(x_j)$ then
    calculate the Hamming's distance $r_{ij} = d(x_i, x_j)$;
    calculate the attractiveness using Equation 20;
    for $k = 1 : length(x_i)$ do
        if $x_{ik} = x_{jk}$ then
            $x(new)_{ik} = x_{jk}$;
        else
            generate a random number $r \sim U(0, 1)$;
            if $r < \beta(r_{ij})$ then
                $x(new)_{ik} = x_{jk}$;
            else
                $x(new)_{ik} = x_{ik}$;
            end
        end
    end
end
return $x(new)_i$;
```

For the next step, we choose α value in the range of [0, 1] and randomise the movement of the firefly for continuous optimisation problems. If the α value is high then x_i will take large step and solution will be far away from the current solution. Similarly, for low α value, the new solution will be within the small neighborhood of the current solution. So α value determines the condition for global and local search and an appropriate value of α determines the balance between global search and local search procedure. But obtaining the exact level of α is not an easy task. For binary variables, this α-step only represents the change in bit values for a particular firefly. There are two ways by which we may change in bit values for binary variables: flip and swap procedure. One can use any of these two procedures as the α-step for solving knapsack problem. The term α represents the probability that a particular bit will change or not. The number of bits (nB) for which there is a change in bit value depends on Hamming distance (i.e., r_{ij}). Because here our aim is to minimize the distance between firefly i and j, as firefly x_j is brighter than firefly x_i; so x_i attracts towards x_j. Therefore, $nB = \alpha \times r_{ij}$ and α should be small; otherwise the Hamming distance will increase between x_i and x_j rather than decreasing. It is similar to variable distance move (k-distance or k-exchange), and here we choose $k = 1$.

4.5 Generating new firefly

In the original FA, we generate a solution randomly if there is no better solution than a particular one. Opposition-based learning (OBL) introduced by Tizhoosh [55] is used to generate the new solution. It is proved that an opposite solution has a higher chance to be closer to the global optimal than a random solution [49]. Opposite point is defined as

$$\acute{x}_j = a_j + b_j - x_j, \tag{22}$$

where $X = (x_1, x_2, ..., x_n)$ be a point in n-dimensional space and $x_j \in [a_j, b_j], \quad j \in 1, 2, .., n$ [48]. The generalized OBL (GOBL) is defined as

$$\acute{x} = r(a + b) - x, \tag{23}$$

where r is a real number [57]. Here we consider r as a random variable, specifically $r \sim U(0, 1)$. It will transform the solution to a new search space and provide more chances to find better solutions.

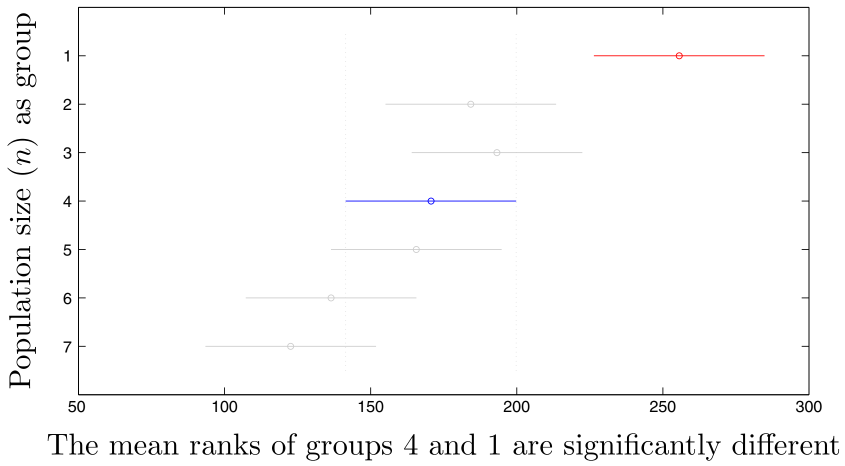

Fig. 3 Multiple comparison of mean ranks of population size (n) for MKP instance f_4

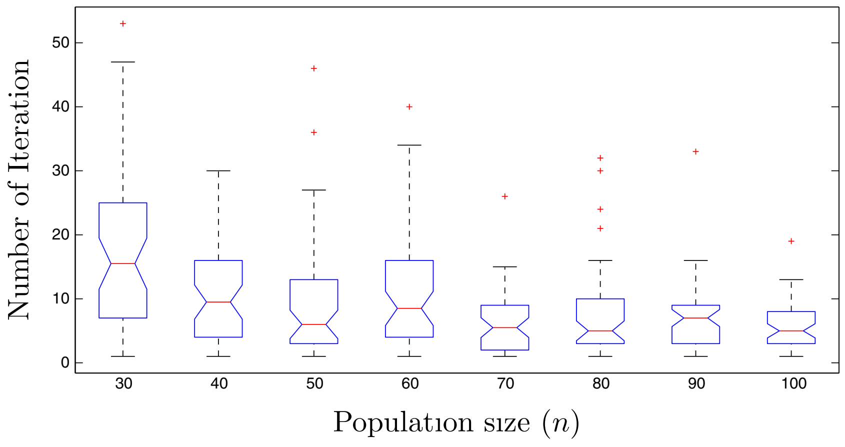

Fig. 4 The effect of population size n for MKP instance f_4

4.6 Repairing infeasible solutions

When the binary string violates the constraint of the given problem, then repair algorithm is employed to make infeasible solutions to feasible one.

For 01 KP, the greedy repair procedure is applied as mentioned earlier. All items in the knapsack are sorted in the decreasing order of their profit to weight ratios. The selection process always chooses the last item for deletion.

For 01 MKP, we use the repair operator given in Algorithm 4 suggested in [15].

5 Experimental results

The proposed BCSA and BFA procedures are coded in Python and implemented on a computer with Intel Core2 Duo CPU E8400 @3.00 GHz with 2GB of RAM in Unix environment. Experiments are also conducted with state-of-art particle swarm optimisation algorithm as a benchmark procedure. Knapsack problems are also solved by many researchers with PSO algorithm. Kong and Tian [36] applied PSO algorithm to solve knapsack problems with the sigmoid function for binary transformation. They used repair operator instead of penalty function and set the number of particles equal to the number of objects of the problem (problem size) for solving multidimensional knapsack problem instances. Tan [54] utilizes the penalty function method and mutate particle position at a specific probability of occurrence, a hybrid technique for solving MKP type problem. Lee et al. [37] proposed GPMBPSO to solve multi-modal benchmark functions. They adopted the concepts of genotype-phenotype representation and the mutation operator of genetic algorithms. Bansal and Deep [5] improved the basic BPSO by introducing a new probability function which maintains the diversity in the swarm and makes it more explorative for solving knapsack and multidimensional knapsack problem instances. Chih et al. [14] introduce the BPSO with time-varying acceleration coefficients and the chaotic BPSO with time-varying acceleration coefficients to solve MKP instances.

Here we utilise the advantages of each algorithm presented by Kong and Tian [36] and Tan [54], and the presented algorithm is the hybrid PSO with repair operator and GA-based mutation operator. For large problem class population size become huge in [36] and it is relatively small ($P = 10$) in [54], so here population size for PSO is set to 40 by the design of experiments (DOE). Results of this hybrid PSO (HPSO) are also shown for comparison along with BPSO [33], GPMBPSO [37], and MBPSO [5] for KP instances. On the other hand, MKP instances are compared with MBPSO [5], CBPSO [16], BPSOTVAC [14], and CBPSOTVAC [14]. For large MKP instance comparison is provided with other two highly efficient algorithms NGHS introduced by Zou et al. [69] and ABHS by Wang et al. [58].

5.1 Benchmark problem set

Two sets of standard benchmark problem instances for knapsack problems are considered in our study. The first set contains ten instances taken from different literature. The

Table 3 Kruskal-Wallis ANOVA table

Source	SS	df	MS	χ^2	$p > \chi^2$
Groups	567223.71	7	81031.96	42.61	3.96E-07
Error	4743749.79	392	12101.40		
Total	5310973.5	399			

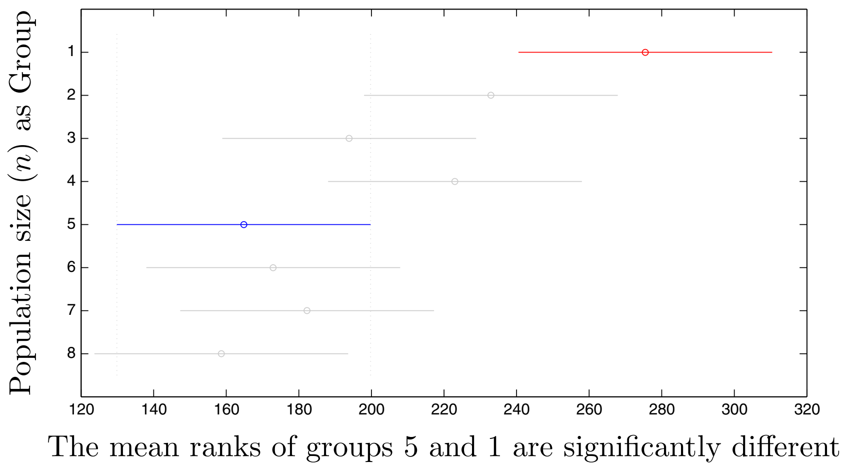

Fig. 5 Multiple comparison of mean ranks of population size (n) for MKP instance f_4

number of items in these instances is ranging from 4 to 23. Detailed information is provided in [8]. The other set of benchmark problem instances is taken from [5], which contains total 25 instances. Bansal and Deep [5] proposed the modified binary PSO algorithm to solve these test problem instances. Problem instances are ranging between 8 and 24.

Similar to KP instances, the performance of different algorithms are extensively investigated by a large number of experimental studies conducted over a set of problem instances collected from the literature.We select benchmark problem instances of 01 MKP from OR-Library [7]. Total seven instances are investigated corresponding to small class; items are ranging from 6 to 50, and the number of the knapsack is 5 or 10. Other 49 instances of medium problem class are also taken from OR-Library. Among medium class problem instances, Senyu and Toyada [50] introduced SENTO class, Weingartner and Ness [60] introduced WEING class, Shi [52] introduced WEISH class, and Freville and Plateau [25] introduced PB and HP classes.

5.2 Parameter settings of BCSA

There are only two main parameters of CSA, population size (n) and endanger probability (p_a). For solving knapsack type problems, we transform the continuous variables to binary type; and the effects of different parameter settings are evaluated and discussed in this section.

5.2.1 Effect of binary variables

Here, three different types of functions are used for handling binary variables as mentioned in Section 3. Effect of binary variables is shown in Table 1. Experimental setup is done with population size $n = 25$ and $p_a = 0.25$. For every instance, we consider total 50 independent runs; and best, worst solutions among these runs are reported along with average, median and standard deviation from all solutions for problems of SET 1. Also for each instance, we reported the average total time required for a successful run, and the maximum number of iteration is fixed to 100. In Table 1, for every instance, the first row represents the performance

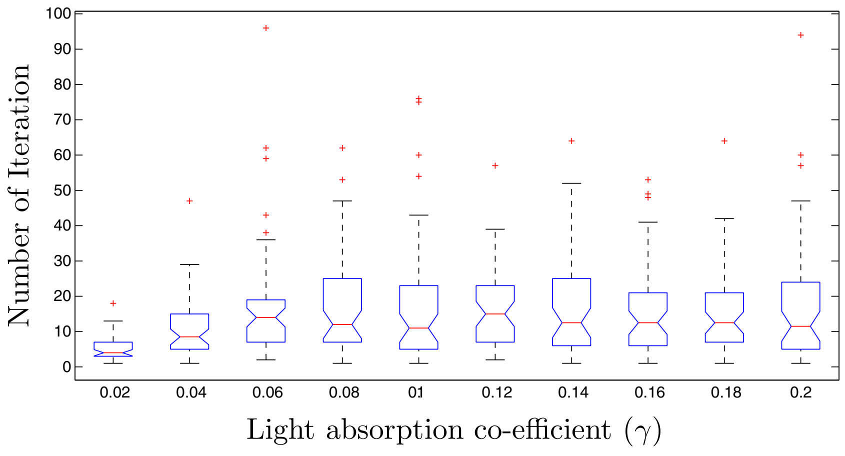

Fig. 6 The effect of light absorption coefficient (γ) for MKP instance f_4

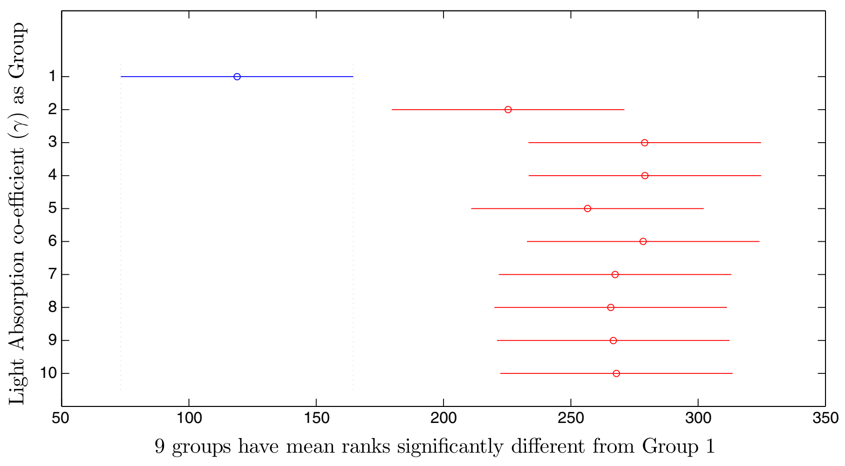

Fig. 7 Multiple comparison of mean ranks of light absorption coefficient (γ) for MKP instance f_4

of the first transformation function, and the performance of the sigmoid transformation is given in the second row, and so on.

From Table 1, it is clear that all transformation functions are effective to solve 01 knapsack instances. However, the sigmoid function gives a much more efficient result, and the solution for the average case is better than any other method. Also, the standard deviation is much low than others.

5.2.2 Effect of endanger probability (p_a)

As suggested in [64], we also inspected the effect of p_a on solving knapsack problems. We fixed the population size $n = 25$ and the maximum number of iterations $iMax = 100$, and used $p_a = \{0.05, 0.1, 0.15, 0.2, 0.25, 0.4, 0.5\}$. The number of iterations required to solve a particular case for different p_a values is shown in Fig. 1 for MKP problem instance f_4. From Kruskal-Wallis test, we get p-value equal to 0.6667, which suggest that we can reject the null hypothesis, and there is no significant difference between sample medians. One can conclude that p_a value does not have any significant effect on solving multidimensional knapsack problem instances.

5.2.3 Effect of population size (n)

Here, we consider different population size for solving MKP instance f_4, with maximum number of iterations $iMax = 100$ and $p_a = 0.2$. Result corresponding to different population sizes $n = \{10, 15, 20, 25, 30, 40, 50\}$ is shown in Fig. 2. Table 2 lists the results of Kruskal-Wallis test. As p-value is very less, so we conclude that at least one sample median is different from the others. Figure 3 shows the results of multiple comparison mean ranks. From the figure, it is clear that Group 1 ($G1 = \{n|n = 10, 15\}$) is significantly different from Group 2 ($G2 = \{n|n = 25, 30, 40, 50\}$). As execution time will increase with very high population size, so we may choose population size $n = 25$.

Table 4 Kruskal-Wallis ANOVA table

Source	SS	df	MS	χ^2	$p > \chi^2$
Groups	1073988.43	9	119332.05	51.54	5.53E-08
Error	9324642.57	490	19029.88		
Total	10398631	499			

5.3 Parameter settings of BFA

As discussed in Section 4, the main parameters of BFA are the population size (n) and the light absorption coefficient (γ). In this section, the effect of these two parameters for solving knapsack problems is discussed.

5.3.1 Effect of population size (n)

For the experimental study we choose population size, $n = \{30, 40, 50, 60, 70, 80, 90, 100\}$ and light absorption coefficient $\gamma = 0.1$. Here we consider total 50 independent runs for each case and the relation between the number of iterations to solve a particular instance and the population size is shown in Fig. 4. Anova table of Kruskal-Wallis test is given in Table 3 and p-value of the corresponding test is $3.96E-07$. Results of multiple comparison mean ranks are shown in Fig. 5. Two groups $G1 = \{n|n = 30, 40\}$ and $G2 = \{n|n = 70, 80, 90, 100\}$ are significantly different from each other. So we may choose population size $n = 70$ for solving multidimensional knapsack problem instances.

5.3.2 Effect of light absorption coefficient (γ)

Here, we consider population size $n = 60$ and light absorption co-efficient $\gamma = \{0.02, 0.04, 0.06, 0.08, 0.09,$

0.10, 0.12, 0.14, 0.16, 0.18, 0.20}. Total 50 independent runs for each case and the relation between the number of iterations required to solve a particular case and the light absorption co-efficient is shown in Fig. 6 for MKP instance f_4. The results of multiple comparison of mean ranks are given in Fig. 7. Anova table of Kruskal-Wallis test is given in Table 4. From the results, one can conclude that light absorption co-efficient $\gamma \leq 0.02$ for solving MKP instances.

5.4 Performance metrics

Three normalised performance measures are used for algorithm's performance analysis. The measures are the solution

Table 5 Performance metrics for 01 KP SET 1 instances

f	best	average	$QMetric$	$SRate$	$SSpeed$
BCSA					
f_1	295	295	1	1	0.9981
f_2	1024	1024	1	1	0.9954
f_3	35	35	1	1	0.9991
f_4	23	23	1	1	0.9990
f_5	481.07	481.07	1	1	0.9994
f_6	52	52	1	1	0.9996
f_7	107	107	1	1	0.9995
f_8	9767	9767	1	1	0.9988
f_9	130	130	1	1	0.9992
f_{10}	1025	1025	1	1	0.9953
BFA					
f_1	295	295	1	1	0.9985
f_2	1024	1023.3	1	0.88	0.8758
f_3	35	35	1	1	0.9985
f_4	23	23	1	1	0.9985
f_5	481.07	481.07	1	1	0.9988
f_6	52	52	1	1	0.9992
f_7	107	107	1	1	0.9991
f_8	9767	9766.3	1	0.94	0.9389
f_9	130	130	1	1	0.9988
f_{10}	1025	1024.5	1	0.92	0.9153
HPSO					
f_1	295	295	1	1	0.9984
f_2	1024	1024	1	1	0.9908
f_3	35	35	1	1	0.9990
f_4	23	23	1	1	0.9990
f_5	481.07	481.07	1	1	0.9994
f_6	52	52	1	1	0.9996
f_7	107	107	1	1	0.9994
f_8	9767	9767	1	1	0.9990
f_9	130	130	1	1	0.9992
f_{10}	1025	1025	1	1	0.9888

Table 6 Performance metrics for 01 MKP SET 1 instances

f	best	average	$QMetric$	$SRate$	$SSpeed$
BCSA					
f_1	3800	3800	1	1	0.9994
f_2	8706.1	8706.1	1	1	0.9978
f_3	4015	4015	1	1	0.9994
f_4	6120	6120	1	1	0.9994
f_5	12400	12400	1	1	0.9994
f_6	10618	10617.72	1	0.98	0.7782
f_7	16537	16537	1	1	0.6965
BFA					
f_1	3800	3800	1	1	0.9990
f_2	8706.1	8687.5	1	0.72	0.7190
f_3	4015	4007.4	1	0.24	0.2398
f_4	6120	6073	1	0.08	0.0799
f_5	12400	12367	1	0.54	0.5397
f_6	10588	10553	0.69	0	0
f_7	16508	16485	0.79	0	0
HPSO					
f_1	3800	3800	1	1	0.9994
f_2	8706.1	8706.1	1	1	0.9990
f_3	4015	4015	1	1	0.9989
f_4	6120	6120	1	1	0.9986
f_5	12400	12400	1	1	0.9987
f_6	10618	10612	1	0.56	0.4296
f_7	16537	16537	1	1	0.9586

quality, the rate of success and the speed of reaching a solution [53]. To compare the rate of success of different algorithms on different problems, each problem/algorithm combination is run using a maximum number of function evaluations ($MaxFES$) as the termination condition. For all problems, the value of $MaxFES$ is set to $10000 \times D$, where D is the dimension of the problem.

The quality of solution obtained by the particular algorithm is quantified by $Qmetric$ and is given by

$$QMetric = 2^{q^{10^2}} - 1; \tag{24}$$

where q is the normalised measure of solution quality [41]. It is defined as,

$$q = \frac{\hat{f} - f^{min}}{op - f^{min}}, \tag{25}$$

where $\hat{f}$ is the maximum value of the objective function found by the algorithm, op is the best known solution and f^{min} is the minimum objective function value for the concerning problem instance (zero).[1] Here we consider that 5 %

[1]Knapsack problem is a maximization problem and decision variable x_j can take two values either zero or one.

Table 7 Optimal parameter settings

Parameter	BCSA	BFA	HPSO
Population size (n)	25	60	40
Probability of mutation (p_m)	na	na	0.06
Probability of discover (p_a)	0.2	na	na
Light absorption co-efficient (γ)	na	0.02	na
Self influence co-efficient (c_1)	na	na	1.496
Swarm influence co-efficient (c_2)	na	na	1.496
Momentum co-efficient (w)	na	na	0.7298
Maximum number of iteration ($iMax$)	$20 \times n$	$20 \times n$	$20 \times n$
Maximum time limit	60	60	60
Maximum number of iteration without improvement	$0.7 \times iMax$	$0.7 \times iMax$	$0.7 \times iMax$

gap or larger from the best known solution is not a significant solution, and $QMetric$ value will be zero (rounded up to 2 decimal places). A particular run is regarded as successful if the run reaches the global optimum before the $MaxFES$ of the problem is exceeded. The success rate ($SRate$) is defined as the ratio of the number of successful

Table 8 Results of problem SET 2 of 01 KP instances

Pb	BCSA		BFA		HPSO		BPSO	GPMBPSO	MBPSO
Ins	best	avg	best	avg	best	avg	avg	avg	avg
8a	3924400	**3924400**	3924400	**3924400**	3924400	3919748.8	3921857.19	3922251.98	**3924400**
8b	3813669	**3813669**	3813669	**3813669**	3813669	3813049.16	3807911.86	3807671.43	**3813669**
8c	3347452	**3347452**	3347452	**3347452**	3347452	**3347452**	3328608.71	3326300.19	**3347452**
8d	4187707	**4187707**	4187707	**4187707**	4187707	**4187707**	4186088.27	4184469.54	**4187707**
8e	4955555	**4955555**	4955555	**4955555**	4955555	**4955555**	4932737.28	4921758.82	4954571.72
12a	5688887	5688757.34	5688887	**5688887**	5688887	5686293.8	5683694.29	5678227.28	5688552.41
12b	6498597	**6498597**	6498597	**6498597**	6498597	6484784.88	6478582.96	6476487.08	6493130.57
12c	5170626	**5170626**	5170626	**5170626**	5170626	5170455	5166957.08	5162237.91	5170493.3
12d	6992404	**6992404**	6992404	**6992404**	6992404	**6992404**	6989842.73	6988151.02	6992144.26
12e	5337472	**5337472**	5337472	**5337472**	5337472	**5337472**	5316879.59	5301119.31	**5337472**
16a	7850983	7840365.4	7850983	**7850983**	7850983	7833489.7	7834900.26	7826923.53	7843073.29
16b	9352998	**9352998**	9352998	**9352998**	9352998	9342132.84	9334408.62	9326158.74	9350353.39
16c	9151147	**9151147**	9151147	**9151147**	9151147	9150126.38	9118837.47	9114581.85	9144118.38
16d	9348889	9339618.52	9348889	**9348889**	9348889	9336318.32	9321705.87	9317336.67	9337915.64
16e	7769117	**7769117**	7769117	**7769117**	7769117	7759149.56	7758572.21	7757247.79	7764131.81
20a	10727049	**10718386.84**	10727049	10725342.6	10727049	10705773.82	10707360.91	10702954.99	10720314.03
20b	9818261	**9818261**	9818261	**9818261**	9818261	9814973.14	9791306.65	9786719.85	9805480.48
20c	10714023	10713094.36	10714023	**10713645.74**	10714023	10712587.86	10703423.34	10695550.75	10710947.05
20d	8929156	8921824.58	8929156	**8929156**	8929156	8911203.8	8910152.57	8905564.36	8923712.21
20e	9357969	**9357969**	9357969	**9357969**	9357969	9357222.96	9349546.98	9343911.1	9355930.35
24a	13549094	**13549094**	13549094	13544867	13549094	13537746.36	13510432.96	13506115.12	13532060.07
24b	12233713	12211698.48	12233713	12208229.7	12233713	12199295.38	12205346.16	12202425.75	**12223442.61**
24c	12448780	**12448780**	12448780	**12448780**	12448780	12447402.54	12427880.56	12419101.82	12443349.03
24d	11815315	11810156.28	11815315	**11810429.92**	11815315	11807623.78	11792064.76	11791581.41	11803712.38
24e	13940099	**13934482.92**	13940099	13933325.08	13940099	13927312	13922797.55	13921046.22	13932526.16

Best results found in corresponding experiments are given in bold emphasis

Table 9 Average Rankings of the algorithms for knapsack problem instances

Algorithm	BCSA	BFA	HPSO	BPSO	GPMBPSO	MBPSO
Ranking	5.06	5.34	3.38	2.12	1.12	3.98

runs and the total number of runs. The success speed of a run r is defined as

$$SSpeed_r = \begin{cases} 0, & \text{if the run is not successful} \\ \frac{MaxFES-(FES_r-1)}{MaxFES}, & \text{otherwise.} \end{cases} \qquad (26)$$

Where FES_r represents the number of function evaluations for the given run r. The success speed over ns successful runs is defined as

$$SSpeed = \begin{cases} \frac{\sum_{r=1}^{ns} SSpeed_r}{ns}, & \text{if } ns > 0 \\ 0, & \text{if } ns = 0. \end{cases} \qquad (27)$$

To illustrate the use of these three performance metrics, Table 5 shows the results of benchmark problem instances of KP (SET 1) for BCSA, BFA and HPSO. All the three algorithms perform well, and $QMetric$ values are one for all the cases. Though BFA finds little bit difficulties to solve three problem instances (f_2, f_8 and f_{10}); $SRate$ and $SSpeed$ are low for these cases.

Table 6 shows the performance metrics of BCSA, BFA and HPSO for 01 MKP instance SET 1. BCSA and HPSO perform well compared to BFA. Among total seven instances, $SSrate$ and $SSpeed$ are substantially low compared to other algorithms for BFA except problem instance f_1. For problem instance f_6 and f_7, BFA is not able to find best-known solution at all, $QMetric$ values are 0.69 and 0.79, respectively. BCSA and HPSO also find it little difficult for these two instances. $SSRate$ and $SSpeed$ of HPSO are low for function f_6, and $SSpeed$ of BCSA is slightly low for function f_7.

5.5 Comparative study

Here we consider the second set of problem instances for both KP and MKP. Experiments are conducted to compare proposed three algorithms with BPSO [33], GPMBPSO [37], and MBPSO [5]. An optimal parameter setting for different algorithms is shown in Table 7.

Total 50 independent trials are conducted for each problem instance. Best and average results found out between these runs are reported in Table 8 for knapsack problem instances. Non-parametric tests are used for comparing algorithms [26–28, 40]. The average ranking of Friedman

Table 10 Adjusted p-values for SET 2 problem instances of KP

No	Hypothesis	unadjusted p	p_{Neme}	p_{Holm}	p_{Shaf}	p_{Berg}
1	BFA vs GPMBPSO	1.52E-15	**2.28E-14**	**2.28E-14**	**2.28E-14**	**2.28E-14**
2	BCSA vs GPMBPSO	9.63E-14	**1.44E-12**	**1.35E-12**	**9.63E-13**	**9.63E-13**
3	BFA vs BPSO	1.16E-09	**1.74E-08**	**1.51E-08**	**1.16E-08**	**1.16E-08**
4	BCSA vs BPSO	2.76E-08	**4.14E-07**	**3.31E-07**	**2.76E-07**	**1.66E-07**
5	GPMBPSO vs MBPSO	6.48E-08	**9.73E-07**	**7.13E-07**	**6.48E-07**	**4.54E-07**
6	HPSO vs GPMBPSO	1.95E-05	**2.92E-04**	**1.95E-04**	**1.95E-04**	**1.17E-04**
7	BFA vs HPSO	2.12E-04	**3.18E-03**	**1.91E-03**	**1.49E-03**	**1.49E-03**
8	BPSO vs MBPSO	4.40E-4	**6.59E-03**	**3.52E-03**	**3.08E-03**	**1.76E-03**
9	BCSA vs HPSO	1.50E-03	**2.24E-02**	**1.05E-02**	**1.05E-02**	**6.00E-03**
10	BFA vs MBPSO	1.02E-02	0.15	0.06	0.06	**0.04**
11	HPSO vs BPSO	0.02	0.26	0.09	0.07	**0.04**
12	BCSA vs MBPSO	0.04	0.62	0.17	0.17	0.08
13	BPSO vs GPMBPSO	0.06	0.88	0.18	0.18	0.18
14	HPSO vs MBPSO	0.26	3.85	0.51	0.51	0.51
15	BCSA vs BFA	0.60	8.95	0.60	0.60	0.60

Best results found in corresponding experiments are given in bold emphasis

Table 11 Results of problem SET 2 of 01 MKP instances

Problem	BCSA		BFA		HPSO		Al-I	Al-II	Al-III	Al-IV
Instances	*best*	*avg*	*best*	*avg*	*best*	*avg*	*MAD*	*MAD*	*MAD*	*MAD*
WEING1e	**141278**	**141278**	**141278**	**141278**	**141278**	**141278**	–	–	–	–
SENTO1	**7772**	7770.9	**7772**	7760.4	**7772**	7761.2	44.81	198.29	8.74	136.28
SENTO2	**8722**	**8722**	**8722**	8691.9	**8722**	**8722**	24.85	103.32	9.42	53.53
WEING1	**141278**	**141278**	**141278**	**141278**	**141278**	**141278**	110.79	**0**	**0**	51.25
WEING2	**130883**	**130883**	**130883**	**130883**	**130883**	**130883**	117.45	**0**	**0**	123.19
WEING3	**95677**	95673	**95677**	95665	**95677**	95629	1053.2	**0**	6.42	173.07
WEING4	**119337**	**119337**	**119337**	**119337**	**119337**	**119337**	570.6	**0**	**0**	42.83
WEING5	**98796**	**98796**	**98796**	**98796**	**98796**	**98796**	1629.21	**0**	**0**	85.62
WEING6	**130623**	**130623**	**130623**	**130623**	**130623**	**130623**	310.2	**0**	11.7	91.71
WEING7	**1095445**	1095384.52	1095382	1095382	1095382	1095371.44	660.86	32690.6	281.23	11272.9
WEING8	**624319**	**624319**	**624319**	624242.14	**624319**	**624319**	5824.7	118166	1872.44	27128.4
WEISH01	**4554**	**4554**	**4554**	**4554**	**4554**	**4554**	10.9	**0**	**0**	5.45
WEISH02	**4536**	4533.8	**4536**	**4536**	**4536**	4535.7	8.39	1.05	1.8	4.12
WEISH03	**4115**	**4115**	**4115**	**4115**	**4115**	**4115**	20.54	**0**	0.63	9.21
WEISH04	**4561**	**4561**	**4561**	**4561**	**4561**	**4561**	1.76	**0**	**0**	8.59
WEISH05	**4514**	**4514**	**4514**	**4514**	**4514**	**4514**	0.54	**0**	**0**	8.11
WEISH06	**5557**	**5557**	**5557**	**5557**	**5557**	**5557**	15.36	5.47	6.68	23.21
WEISH07	**5567**	**5567**	**5567**	**5567**	**5567**	**5567**	10.2	3.45	0.7	19.17
WEISH08	**5605**	5604.9	**5605**	5603.9	**5605**	5603.6	7.24	0.72	0.42	8.84
WEISH09	**5246**	**5246**	**5246**	**5246**	**5246**	**5246**	10.61	1.36	**0**	13.01
WEISH10	**6339**	**6339**	**6339**	6338.4	**6339**	**6339**	10.84	42.57	1.43	57.16
WEISH11	**5643**	**5643**	**5643**	**5643**	**5643**	**5643**	29.48	79.9	7.42	110.85
WEISH12	**6339**	**6339**	**6339**	6338.4	**6339**	**6339**	16.35	57.3	0.29	107.5
WEISH13	**6159**	**6159**	**6159**	**6159**	**6159**	**6159**	8.47	59.33	**0**	38.62
WEISH14	**6954**	**6954**	**6954**	6952	**6954**	**6954**	16.09	210.47	0.62	116.23
WEISH15	**7486**	**7486**	**7486**	7465.5	**7486**	7481.5	10.55	193.51	**0**	161.45
WEISH16	**7289**	**7289**	**7289**	7267.6	**7289**	7287.5	7.66	139.24	1.16	143.29
WEISH17	**8633**	**8633**	**8633**	8605.8	**8633**	**8633**	5.76	91.8	**0**	85.29
WEISH18	**9580**	**9580**	**9580**	9451	**9580**	9579.3	14.65	259.34	2.79	99.14
WEISH19	**7698**	**7698**	**7698**	7692.3	**7698**	**7698**	20.83	429.39	4.9	169.45
WEISH20	**9450**	**9450**	9445	9385.7	**9450**	9449.5	10.81	336.41	3.78	117.89
WEISH21	**9074**	**9074**	**9074**	9042.1	**9074**	9066.5	17.85	347.65	6.06	125.78
WEISH22	**8947**	8945.6	8926	8843.1	**8947**	8943.8	29.73	674.21	15.12	172.8
WEISH23	**8344**	8342.7	**8344**	8306.9	**8344**	8343.4	29.65	670.43	1.11	179
WEISH24	**10220**	**10220**	10159	10047	**10220**	10205	17.48	367.36	3.04	113.72
WEISH25	**9939**	9930	9906	9830.1	9936	9930.8	15.13	449.77	4.54	112.43
WEISH26	**9584**	**9584**	9552	9441.5	**9584**	9578.9	27.32	895.39	11.44	270.13
WEISH27	**9819**	**9819**	9757	9637.3	**9819**	**9819**	23.7	967.43	0.39	211.46
WEISH28	**9492**	**9492**	9469	9373	**9492**	**9492**	15.21	980.45	2.99	368.74
WEISH29	**9410**	**9410**	**9410**	9303.9	**9410**	**9410**	26.73	981.44	3.19	384.5
WEISH30	**11191**	**11191**	11126	11026	**11191**	**11191**	11.6	548.1	0.52	203.79
PB1	**3090**	3080.2	**3090**	3079.7	**3090**	3086.1	32.62	5.93	9	10.26
PB2	**3186**	3174.8	**3186**	3164.6	**3186**	3167.6	44.69	4.28	4.5	14.45
PB4	**95168**	**95168**	**95168**	**95168**	**95168**	**95168**	2639.8	2.03	228.1	304.33
PB5	**2139**	2138.7	**2139**	**2139**	**2139**	**2139**	49.42	0.17	2.72	3.4
PB6	**776**	764.52	**776**	**776**	**776**	767.64	27.36	8.47	8.7	17.74
PB7	**1035**	**1035**	**1035**	**1035**	**1035**	**1035**	19.89	5.64	5.43	13.05
HP1	**3418**	3407.9	**3418**	3408	**3418**	3410.5	37.52	5.5	11.44	14.1
HP2	**3186**	**3168**	**3186**	3164	**3186**	3160	46.22	4.58	6.51	12.39

Best results found in corresponding experiments are given in bold emphasis

Table 12 Average Rankings of the algorithms for SET 2 MKP instances

Algorithm	BCSA	BFA	HPSO	MPSO	CBPSO	BPSOTVAC	CBPSOTVAC
Ranking	5.72	4.30	5.59	2.52	3.04	4.89	1.94

test is conducted to compare the performance of multiple algorithms on knapsack problem instances. Friedman test statistic value is 97.37 with 5 degrees of freedom, and the corresponding p-value is 5.62E-11 which indicates that one can not reject the null hypothesis and the performance of all the algorithms are not same. Table 9 shows the average ranking of BCSA, BFA, HPSO, BPSO, GPMBPSO and MBPSO on knapsack problem instances of SET 2.

Nemenyi's, Holm's, Shaffer's and Bergemann-Hommel's tests are conducted to investigate the difference between the performance of two different algorithms. Adjusted p-values on pairwise comparisons of all algorithms are reported in Table 10. The null hypothesis is defined as the two algorithms are equivalent for these nonparametric tests. If the null hypothesis is rejected, then there is a significant difference between the performances of these two algorithms. In this work, we have considered the significance level α is 0.05. From the results of Table 10, it is clear that Nemenyi's, Shaffer's and Bergmann-Hommel's procedures reject Hypotheses 1–9. According to Holm's test, we can reject Hypotheses 1–11.

From the results, one can conclude that BCSA and BFA are significantly better than others (MBPSO, HPSO, GPMBPSO and BPSO) for solving knapsack problem instances. There is no significant difference between the performance of BCSA and BFA. MBPSO is significantly better than GPMBPSO and BPSO. There is no significant difference between MBPSO and HPSO. The performance of six algorithms can be sorted by average ranking into the following order: BFA, BCSA, MBPSO, HPSO, BPSO and GPMBPSO. It means that BFA and GPMBPSO are the best and worst ones among these six algorithms, respectively.

Next, we provide the computation study on 01MKP benchmark instances of SET 2 in Table 11. The best solution and average solution among 50 independent runs of BCSA, BFA and HPSO algorithms are reported for each problem instances. For comparing the performance of these algorithms, Friedman test is conducted. Mean average

Table 13 Adjusted p-values

No	Hypothesis	unadjusted p	p_{Neme}	p_{Holm}	p_{Shaf}	p_{Berg}
1	BCSA vs CBPSOTVAC	9.90E-18	**2.08E-16**	**2.08E-16**	**2.08E-16**	**2.08E-16**
2	HPSO vs CBPSOTVAC	1.12E-16	**2.35E-15**	**2.23E-15**	**1.68E-15**	**1.68E-15**
3	BCSA vs MBPSO	4.10E-13	**8.61E-12**	**7.79E-12**	**6.15E-12**	**6.15E-12**
4	HPSO vs MBPSO	3.20E-12	**6.72E-11**	**5.76E-11**	**4.80E-11**	**3.20E-11**
5	BPSOTVAC vs CBPSOTVAC	2.31E-11	**4.84E-10**	**3.92E-10**	**3.46E-10**	**2.54E-10**
6	BCSA vs CBPSO	1.27E-9	**2.67E-8**	**2.03E-8**	**1.91E-8**	**1.40E-8**
7	HPSO vs CBPSO	7.14E-9	**1.50E-7**	**1.07E-7**	**1.07E-7**	**5.00E-8**
8	BFA vs CBPSOTVAC	8.21E-8	**1.72E-6**	**1.15E-6**	**9.04E-7**	**7.39E-7**
9	MBPSO vs BPSOTVAC	8.21E-8	**1.72E-6**	**1.15E-6**	**9.04E-7**	**7.39E-7**
10	CBPSO vs BPSOTVAC	2.90E-5	**6.09E-4**	**3.48E-4**	**3.19E-4**	**1.45E-4**
11	BFA vs MBPSO	5.36E-5	**1.12E-3**	**5.89E-4**	**5.89E-4**	**3.21E-4**
12	BCSA vs BFA	1.31E-3	**0.03**	**0.01**	**0.01**	**0.01**
13	BFA vs HPSO	3.40E-3	0.07	**0.03**	**0.03**	**0.02**
14	BFA vs CBPSO	4.26E-3	0.09	**0.03**	**0.03**	**0.02**
15	CBPSO vs CBPSOTVAC	0.01	0.26	0.09	0.09	0.08
16	BCSA vs BPSOTVAC	0.06	1.23	0.35	0.35	0.24
17	HPSO vs BPSOTVAC	0.11	2.27	0.54	0.54	0.24
18	MBPSO vs CBPSOTVAC	0.19	3.90	0.74	0.74	0.56
19	BFA vs BPSOTVAC	0.19	3.90	0.74	0.74	0.56
20	MBPSO vs CBPSO	0.24	4.99	0.74	0.74	0.56
21	BCSA vs HPSO	0.78	16.31	0.78	0.78	0.78

Best results found in corresponding experiments are given in bold emphasis

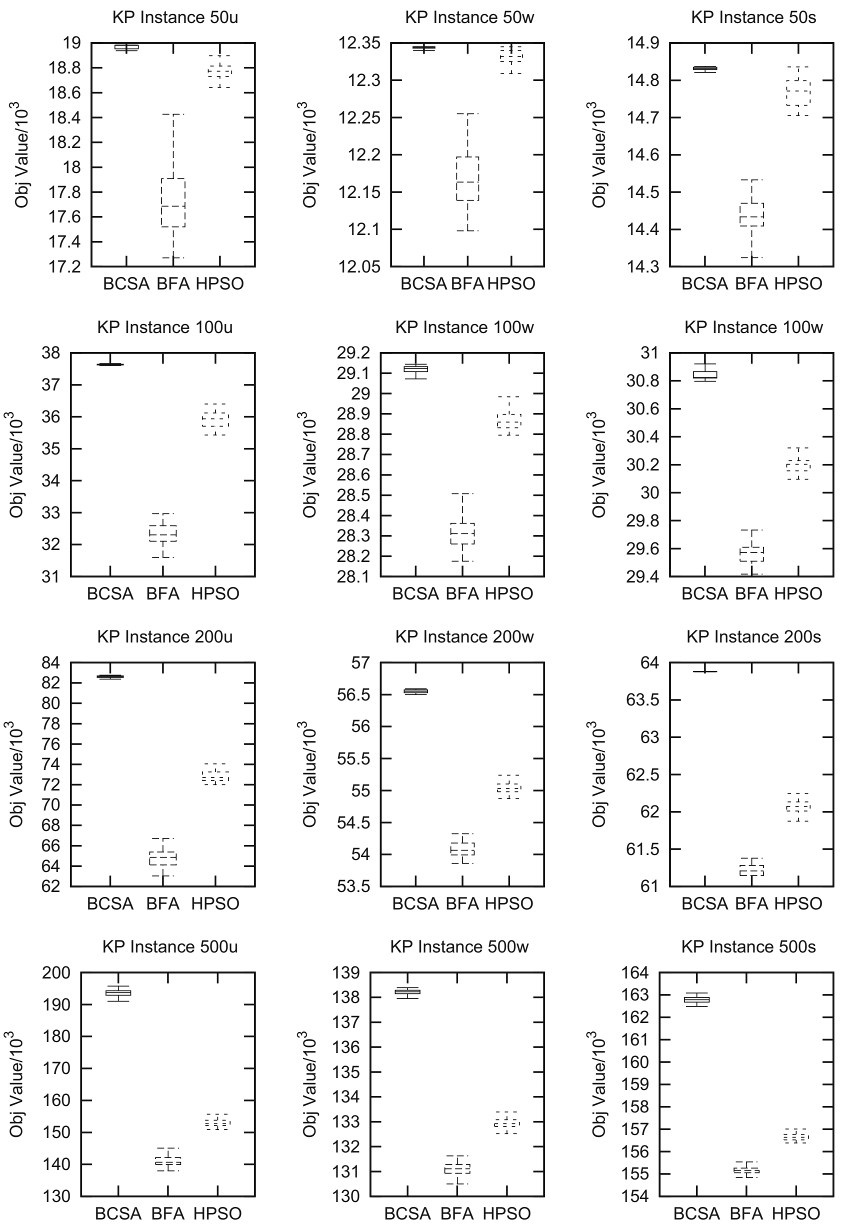

Fig. 8 Solution of medium and large scale KP instances

deviation for other four algorithms Al-I (MBPSO [5]), Al-II (CBPSO [16]), Al-III and Al-IV (BPSOTVAC and CBPSOTVAC [14]) are also reported in the table. The test statistics value is 141.22 with degrees of freedom 6 and corresponding p-value is $8.63E-11$. So one can reject the null hypothesis and conclude that the performance of at least one algorithm is significantly different than other. Table 12 shows the average ranking of BCSA, BFA, HPSO, MBPSO, CBPSO, BPSOTVAC and CBPSOTVAC algorithm on MKP instances of SET 2.

Adjusted p-values of different nonparametric tests (Nemenyi, Holm, Shaffer and Bergmann-Hommel test) are given in Table 13. According to these tests, two algorithms are equivalent under the null hypothesis. Here we have considered significance level $\alpha = 0.05$, and the null hypothesis is rejected if the p-value is less than the α value. According to all the four tests (Nemenyi's, Holm, Shaffer and Bergmann-Hommel test), we can reject Hypotheses 1–12. If we consider only Holm, Shaffer and Bergmann-Hommel test, we can reject Hypothesis 1–14.

From the results, one can infer that HPSO and BCSA are significantly better than CBPSOTVAC, MBPSO, CBPSO and BFA for solving MKP instances; BPSOTVAC is significantly better than CBPSOTVAC, CBPSO and MBPSO; BFA is significantly better than CBPSOTVAC, MBPSO and CBPSO. Again there is no significant difference between the performance of HPSO, BCSA and BPSOTVAC. If the performance of these algorithms is

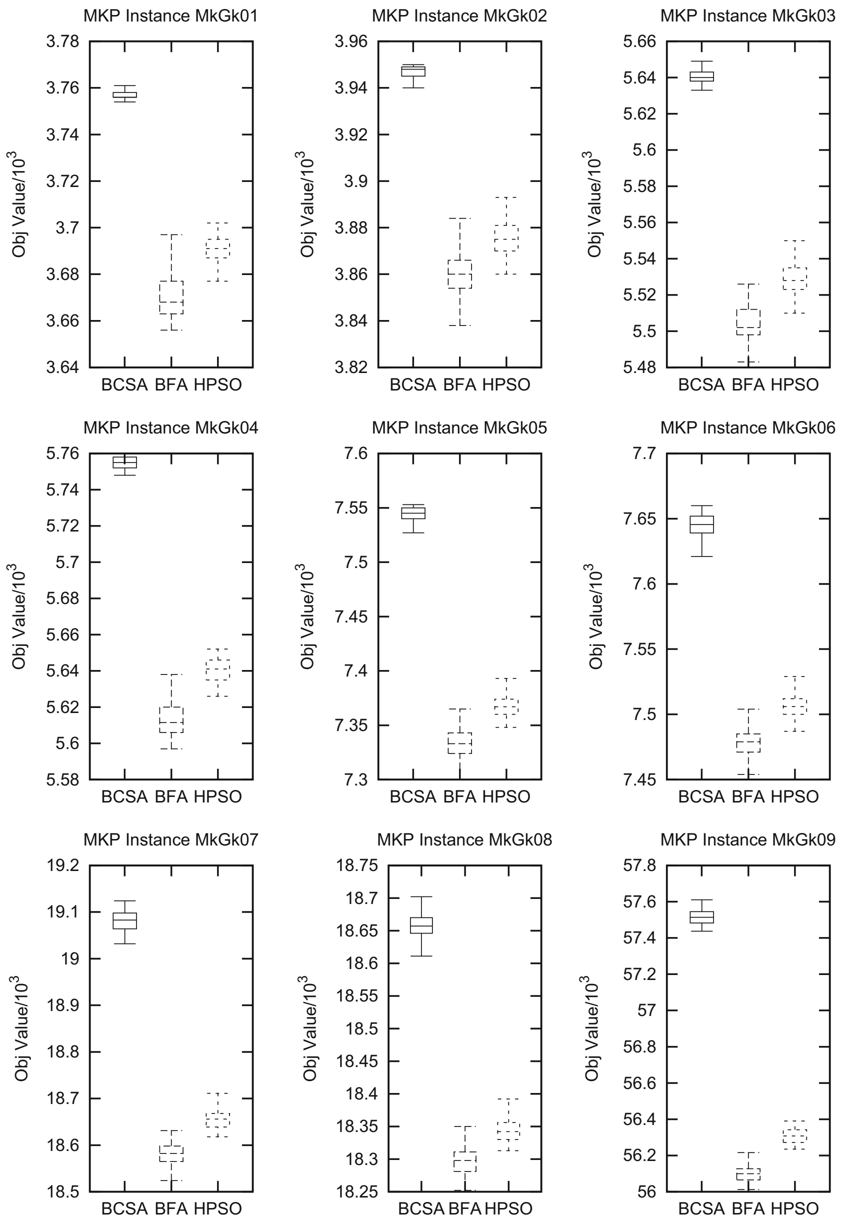

Fig. 9 Solution of large scale MKP instances

sorted by the average ranking, then we get the following order: BCSA, HPSO, BPSOTVAC, BFA, CBPSO, MBPSO and CBPSOTVAC; where BCSA and CBPSOTVAC are the best and worst ones among these seven algorithms, respectively.

5.6 Solution of large class problem instances

Experiments are also conducted on large class problem instances. For KP we choose total twelve instances, generated randomly with average knapsack capacity. Three

Table 14 Adjusted p-values

No	hypothesis	unadjusted p	p_{Neme}	p_{Holm}	p_{Shaf}	p_{Berg}
1	BCSA vs BFA	9.63E-07	**2.89E-06**	**2.89E-06**	**2.89E-06**	**2.89E-06**
2	BCSA vs HPSO	0.01	**0.04**	**0.03**	**0.01**	**0.01**
3	BFA vs HPSO	0.01	**0.04**	**0.03**	**0.01**	**0.01**

Best results found in corresponding experiments are given in bold emphasis

Table 15 Comparison with other algorithms

Problem	BCSA		BFA		HPSO		NGHS		ABHS	
Instance	Ave.Dev	Var.Dev	Ave.Dev	Var.Dev	Ave.Dev	Var.Dev	Ave.Dev	Var.Dev	Ave.Dev	Var.Dev
MKGk01	**0.230**	0.088	2.548	0.267	2.006	0.157	0.349	0.175	0.270	0.046
MKGk02	0.274	0.081	2.469	0.239	2.053	0.252	0.292	0.093	**0.211**	0.091
MKGk03	**0.166**	0.071	2.574	0.192	2.139	0.155	0.362	0.145	0.195	0.089
MKGk04	**0.152**	0.054	2.607	0.168	2.124	0.159	0.525	0.308	0.216	0.144
MKGk05	**0.167**	0.084	2.957	0.197	2.493	0.143	0.445	0.138	0.236	0.104
MKGk06	0.350	0.121	2.503	0.180	2.152	0.125	0.510	0.171	**0.289**	0.159
MKGk07	0.695	0.115	3.292	0.147	2.899	0.136	0.508	0.136	**0.401**	0.106
MKGk08	0.767	0.109	2.665	0.129	2.433	0.097	0.665	0.135	**0.532**	0.090
MKGk09	0.976	0.079	3.418	0.089	3.052	0.099	0.650	0.105	**0.569**	0.075

Best results found in corresponding experiments are given in bold emphasis

types of problem instances are produced. For the first category the profit and weight vectors are uncorrelated to each other; in the second case these are weakly correlated, and weight and profit vectors are strongly correlated for the last situation. These three types of problem instances are represented by the number objects followed by the correlation structure, like uncorrelated ("u"), weakly correlated ("w") and strongly correlated ("s"), respectively. MKP instances are taken from http://www.cs.nott.ac.uk/~jqd/mkp/index.html, introduced by Glover and Kochenberger [29]. Solutions of BCSA, BFA and HPSO, are shown in Fig. 8 for KP instances, and Fig. 9 presents the solutions for MKP instances.

As viewed from the Fig. 8, for medium problem size ($n = 50$) there is a small difference between the performance levels of three algorithms. But as problem size increases, there is a much more difference between the performance of these three algorithms. BCSA outperforms other two algorithms for all the cases. For comparing the results, the average ranking of Friedman test is conducted. The test statistic value is 24.00 with 2 degrees of freedom, and the corresponding p-value is 6.14E-06. So one can not reject the null hypothesis and suggests that the performance of all the algorithms are not same. The average ranking of BCSA, BFA, and HPSO are 3.0, 1.0, and 2.0 respectively. Adjusted p-values for Nemenyi's, Holm's, Shaffer's and Bergemann-Hommel's tests on pairwise comparisons of all three algorithms are reported in Table 14. Adjusted p-values for all the tests are less than the significance level. Therefore, we can conclude that BCSA is significantly better than BFA and HPSO; and HPSO is significantly better than BFA for solving KP large problem instances. Order ranking of these algorithms is BCSA, HPSO and BFA, where BCSA and BFA are the best and worst ones among these algorithms, respectively.

For MKP instances of the large case, the performance of BCSA, BFA and HPSO are compared with other two algorithms in Table 15; NGHS [69] and ABHS [58]. For comparing the results, the average ranking of Friedman test is conducted. The test statistic value is 31.56 with 4 degrees of freedom, and the corresponding p-value is

Table 16 Adjusted p-values

No	hypothesis	unadjusted p	p_{Neme}	p_{Holm}	p_{Shaf}	p_{Berg}
1	BFA vs ABHS	1.84E-6	**1.84E-5**	**1.84E-5**	**1.84E-5**	**1.84E-5**
2	BCSA vs BFA	2.99E-5	**2.99E-4**	**2.69E-4**	**1.80E-4**	**1.80E-4**
3	HPSO vs ABHS	6.07E-4	**6.07E-3**	**4.85E-3**	**3.64E-3**	**3.64E-3**
4	BFA vs NGHS	1.75E-3	**0.02**	**0.01**	**0.01**	**6.98E-3**
5	BCSA vs HPSO	4.62E-3	**0.05**	**0.03**	**0.03**	**0.01**
6	HPSO vs NGHS	0.07	0.74	0.39	0.29	0.15
7	NGHS vs ABHS	0.10	1.01	0.40	0.40	0.40
8	BFA vs HPSO	0.18	1.80	0.54	0.54	0.40
9	BCSA vs NGHS	0.30	2.97	0.59	0.59	0.40
10	BCSA vs ABHS	0.55	5.51	0.59	0.59	0.55

Best results found in corresponding experiments are given in bold emphasis

Table 17 Comparison with other algorithms for small and medium size instances

Problem	BCSA		BFA		HPSO		NGHS		ABHS	
Instances	AVG	SR	AVG	SR	AVG	SR	AVG	SR	AVG	SR
Sent01	7770.9	90	7760.4	14	7761.2	2	**7772**	**100**	**7772**	**100**
Sent02	**8722**	**100**	8691.9	20	**8722**	**100**	**8722**	**100**	**8722**	**100**
Weing7	1095384.52	0	1095382	0	1095371.44	0	**1095389.5**	**5**	1095382	0
Weing8	**624319**	**100**	624242.14	68	**624319**	**100**	622540.9	45	**624319**	**100**
Knap15	**4015**	**100**	4007.4	24	**4015**	**100**	**4015**	**100**	**4015**	**100**
Knap20	**6120**	**100**	6073	8	**6120**	**100**	**6120**	**100**	**6120**	**100**
Knap28	**12400**	**100**	12367	54	**12400**	**100**	**12400**	**100**	**12400**	**100**
Knap39	**10617.72**	**98**	10553	0	10612	56	10603.1	5	10608.2	30

Best results found in corresponding experiments are given in bold emphasis

2.36E-06. Therefore, one can not reject the null hypothesis; which suggest that the performance of all the algorithms are not same. The average ranking of BCSA, BFA, HPSO, NGHS, and ABHS are 4.11, 1.0, 2.0, 3.33, and 4.56, respectively. Adjusted p-values for Nemenyi's, Holm's, Shaffer's and Bergemann-Hommel's tests on pairwise comparisons of all five algorithms are reported in Table 16. For all the tests null hypothesis is rejected for Hypotheses 1-5. According to the results, ABHS is significantly better than HPSO and BFA; NGHS is significantly better than BFA; BCSA is significantly better than HPSO and BFA. But there is no significant difference between ABHS, NGHS and BCSA. Order ranking of these algorithms is ABHS, BCSA, NGHS, HPSO and BFA, where ABHS and BFA are the best and worst ones among these algorithms, respectively.

Table 17 shows the results for other knapsack instances of small and medium size problem along with the results of other two standard algorithms (NGHS and ABHS). The average of 50 individual runs and corresponding success rate are shown for each case. BCSA performs well for each case except the problem instance "Weing7". Also, other algorithms find it difficult to solve this particular instance. BCSA outperforms other algorithms in the case of "Knap39" instance, and the success rate is very high compared to other algorithms. HPSO performs well except the problem instance "Sent01" and the success rate is very low for this particular case. Among these five algorithms, the performance of BFA is worst compared to others.

6 Conclusion

Here we have considered one of the fundamental combinatorial optimisation problems, knapsack problem and the solution process of this problem is intensively investigated in this paper. We propose the binary cuckoo search algorithm for solving knapsack problems. The combination of local random walk and the global random walk is utilised for the implementation of modified CSA. It provides the balance between intensification and diversification techniques and an excellent exposure to the algorithm for finding out high-quality solutions. Also, we developed the binary firefly algorithm for the knapsack problem; and in the modified FA, the variable distance move is utilised to perform the local search along with repair algorithm. New solutions are generated from outside of the current search domain employing opposition-based learning to explore the unexploited solution regions. It will ensure that the modified FA will not be trapped within the local optimal points. Total 47 benchmark problem instances of 01 KP and 65 benchmark problem instances of 01 MKP is considered for the experimental study.

Also, we have carried out various tests for proper parameter tuning, and the performances of the modified algorithms have been extensively investigated on these standard benchmark instances. In this paper, we consider the particle swarm optimisation algorithm as the reference technique. Proposed algorithms were compared with recently developed several variants of PSO algorithm. For small and medium class problem instances, BCSA outperformed in most of the cases regarding the solution quality, rate of success and speed of reaching best-known solutions. BFA works efficiently for 01 knapsack problem instances but finds difficulties for multidimensional knapsack problem instances. For large case, BCSA totally outperforms others. For KP instances as the dimension of problem increases, the performance of BFA and HPSO degrade. For MKP instances, BCSA again outperforms others. So for large class problem situations, our choice should be BCSA for solving knapsack problems. Numerous statistical tests also confirm the validity of the proposed methods. In future, these algorithms can be extended to solve other combinatorial problems. However, proper parameter tuning is required before carrying out any performance analysis. Further modification on FA is scheduled as future work for improving the solution quality for the large class problem instances for both single and multidimensional knapsack problem.

References

1. Abraham A, Guo H, Liu H (2006) Swarm intelligence: foundations,perspectives and optimization. Springer
2. Agrawal S, Panda R (2012) An efficient algorithm for gray level image enhancement using cuckoo search. In: Swarm, evolutionary, and memetic computing. Springer, pp 82–89
3. Alsmadi MK (2014) A hybrid firefly algorithm with fuzzy-c mean algorithm for mri brain segmentation. Am J Appl Sci 11(9):1676–1691
4. Arntzen H, Hvattum LM, Lokketangen A (2006) Adaptive memory search for multidemand multidimensional knapsack problems. Comput Oper Res 33(9):2508–2525
5. Bansal JC, Deep K (2012) A modified binary particle swarm optimization for knapsack problems. Applied Mathematics and Computation 218:11,042–11,061
6. Baykasoglu A, Ozsoydan FB (2014) An improved firefly algorithm for solving dynamic multidimensional knapsack problems. Expert Syst Appl 41:3712–3725
7. Beasley JE (1990) Or-library: distributing test problems by electronic mail. Journal of Operational Research Society 41(11):1069–1072
8. Bhattacharjee KK, Sarmah SP (2014) Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. Appl Soft Comput 19:252–263
9. Bhattacharjee KK, Sarmah SP (2015) A binary cuckoo search algorithm for knapsack problems. IEEE, Dubai
10. Bhattacharjee KK, Sarmah SP (2015) A binary firefly algorithm for knapsack problems. In: IEEE international conference on industrial engineering and engineering management. IEEE, Singapore
11. Bonabeau E, Meyer C (2001) Swarm intelligence: a whole new way to think about business. Harv Bus Rev 79(5):106–115
12. Chandrasekaran K, Simon S (2012) Network and realiability constrained unit commitment problem using binary real coded firefly algorithm. Int J Electr Power Energy Syst 43(1):921–932
13. Chandrasekaran K, Simon SP (2012) Multi-objective scheduling problem: hybrid approach using fuzzy assisted cuckoo search algorithm. Swarm Evol Comput 5:1–16
14. Chih M, Lin CJ, Chern MS, Ou TY (2014) Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. Appl Math Model 38:1338–1350
15. Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4(1):63–86
16. Chuang LY, Yang CH, Li JC (2011) Chaotic maps based on binary particle swarm optimization for feature selection. Appl Soft Comput 11:239–248
17. Civicioglu P, Besdok E (2013) A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif Intell Rev 39(4):315–346
18. Durgun I, Yildiz AR (2012) Structural design optimization of vehicle components using cuckoo search algorithm. Material Testing 54(3):185–188
19. Durkota K (2011) Implementation of a discrete firefly algorithm for the qap problem within the seage framework. Master's thesis, Electrical Engineering, Czech Technical University, Prague
20. Egeblad J, Pisinger D (2009) Heuristic approaches for the two- and three-dimensional knapsack packing problem. Comput Oper Res 36(4):1026–1049
21. Elkeran A (2013) A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. Eur J Oper Res 231(3):757–769
22. Engelbrecht AP (2005) Fundamentals of computational swarm intelligence. Wiley, Chichester
23. Falcon R, Almeida M, Nayak A (2011) Fault identification with binary adaptive fireflies in parallel and distributed systems. In: IEEE congress on evolutionary computation, CEC-2011. IEEE, pp 1359–1366
24. Fister I, Fister Jr. I, Yang XS, Brest J (2013) A comprehensive review of firefly algorithm. Swarm Evol Comput 13:34–46
25. Freville A, Plateau G (1990) Hard 0-1 multiknapsack test problems for size reduction methods. Investigation Operativa 1:251–270
26. Garcia S, Fernandez A, Luengo J (2009) A study of statistical techniques and performance measures for genetic-based machine learning: accuracy and interpretability. Soft Comput 13:959–977
27. Garcia S, Fernandez A, Luengo J, Herrera F (2010) Advanced non-parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf Sci 180:2044–2064
28. Garcia S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms behavior: a case study on the cec2005 special season on real parameter optimization. J Heuristics 15:617–644
29. Glover F, Kochenberger GA (1996) Critical event tabu search for multidimensional knapsack problems. Meta-heuristics: Theory and Applications, Kluwer Academic Publisher, Cited By (since 1996) 63
30. Jati G (2011) Evolutionary discrete firefly algorithm for travelling salesman problem. Adaptive and Intelligent Systems LNAI 6943:393–403
31. Kavousi-Fard A, Samet H, Marzbani F (2014) A new hybrid modified firefly algorithm and support vector regression model for accurate short term load forecasting. Expert Syst Appl 41(13):6047–6056
32. Ke L, Feng Z, Ren Z, Wei X (2010) An ant colony optimization approach for the multidimensional knapsack problem. J Heuristics 16(1):65–83
33. Kenedy J, Eberhart RC (1997) A discrete binary version of the particle swarm optimization. In: Computational cybernatics and simulation, vol 5. IEEE International Conference, Systems, Man, and Cybernatics, pp 4104–4108
34. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: IEEE conference of neural networks, vol 4, pp 1942–1948
35. Khadwilard A, Chansombat S, Thepphakorn T, Thapatsuwan P, Chainate W, Pongcharoen P (2011) Application offirefly algorithm and its parameter setting for job shop scheduling. In: The first symposium on hands-on research and development, pp 1–10
36. Kong M, Tian P (2006) Apply the particle swarm optimization to the multidimensional knapsack problem. Lect Notes Comput Sci LNAI(4029):1140–1149
37. Lee S, Soak S, Oh S, Pedrycz W, Jeon M (2008) Modified binary particle swarm optimization. Prog Nat Sci 18(9):1161–1166
38. Li ZK, Li N (2009) A novel multi-mutation binary particle swarm optimization for 0/1 knapsack problem. In: Control and decision conference 2009, pp 3042–3047
39. Liu Y, Liu C (2009) A schema-guiding evolutionary algorithm for 0-1 knapsack problem. In: 2009 International association of computer science and information technology-spring conference, pp 160–164

40. Luengo J, Garcia S, Herrera F (2009) A study on the use of statistical tests for experimentation with neural networks: analysis of parametric test conditions and non-parametric tests. Expert Syst Appl 36:7798–7808
41. Malan KM, Engelbrecht AP (2014) Recent advances in the theory and application of fitness landscapes. In: Ritcher H, Engelbrecht A (eds) Emergence, complexity and computation, vol 6. Springer, pp 103–132
42. Mantegna RN (1994) Fast, accurate algorithm for numerical simulation of levy stable stochastic processes. Phys Rev E 49(5):4677–4683
43. Mishra A, Agarwal C, Sharma A, Bedi P (2014) Optimized grayscale image watermarking using dwt-svd and firefly algorithm. Expert Syst Appl 41(17):7858–7867
44. Ouaarab A, Ahiod B, Yang XS (2015) Random-key cuckoo search for the travelling salesman problem. Soft Comput 19:1099–1106
45. Palit S, Sinha S, Molla M, Khanra A, Kule M (2011) A cryptoanalytic attack on the knapsack crypto system using binary firefly algorithm. In: The second international conference on computer and communication technology, ICCCT-2011, IEEE, pp 428–432
46. Pirkul H (1987) A heuristic solution procedure for the multiconstraint zero-one knapsack problem. Nav Res Logist 34:161–172
47. Rahmaniani R, Ghaderi A (2013) A combined facility location and network design problem with multi-type of capacited links. Appl Math Model 37:6400–6414
48. Rahnamayan S, Tizhoosh HR, Salama MMA (2006) Opposition-based differential evolution algorithms. In: IEEE congress on evolutionary computation, pp 2010–2017
49. Rahnamayan S, Tizhoosh HR, Salama MMA (2008) Opposition versus randomness in soft computing techniques. Appl Soft Comput 8:906–918
50. Senyu S, Toyada Y (1967) An approach to linear programming with 0-1 variables. Manag Sci 15:B196–B207
51. Shi HX (2006) Solution to 0/1 knapsack problem based on improved ant colony algorithm. In: International conference on information acquisition 2006, pp 1062–1066
52. Shi W (1979) A branch and bound method for the multiconstraint zero one knapsack problem. J Oper Res Soc 30:369–378
53. Suganthan PN, Hasen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S (2005) Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Tech. rep., Nanyang Technical University, Singapore
54. Tan RR (2007) Hybrid evolutionary computation for the development of pollution prevention and control strategies. J Clean Prod 15(10):902–906
55. Tizhoosh HR (2005) Opposition-based learning: a new scheme for machine intelligence. In: International conference on computational intelligence for modeling control and automation, pp 695–701
56. Walton S, Hassan O, Morgan K, Brown M (2011) Modified cuckoo search: a new gradient free optimisation algorithm. Chaos Solitons Fractals 44(9):710–718
57. Wang H, Zhijian W, Rahnamayan S (2011) Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. Soft Comput 15:2127–2140
58. Wang L, Yang RX, Xu Y (2013) An improved adaptive binary harmony search algorithm. Inf Sci 232:58–87
59. Wanga Y, Feng XY, Huang YX, Pub DB, Zhoua WG, Liang YC, Zhou CG (2007) A novel quantum swarm evolutionary algorithm and its applications. Neurocomputing 70:633–640
60. Weingartner HM, Ness DN (1967) Methods for the solution of the multi-dimensional 0/1 knapsack problem. Oper Res 15(1):83–103
61. Yang XS (2008) Firefly algorithm. Nature-Inspired Metaheuristic Algorithms 20:79–90
62. Yang XS (2009) Firefly algorithm for multimodal optimization. Lect Notes Comput Sci 5792:169–178
63. Yang XS (2010) Nature-inspired metaheuristic algorithms. Luniver Press
64. Yang XS, Deb S (2009) Cuckoo search via levy flights. In: World congress on nature biologically inspired computing, pp 210–214
65. Yang XS, Gandomi AH, Talatahari S, Alavi AH (2013) Metaheuristics in water, geotechnical and transport engineering, Elsevier
66. Yildiz AR (2013) Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. Int J Adv Manuf Technol 64(1-4):55–61
67. Zhao JF, Huang TL, Pang F, Liu YJ (2009) Genetic algorithm based on greedy strategy in the 0-1 knapsack problem. In: 3Rd international conference on genetic and evolutionary computing, WGEC ’09, pp 105–107
68. Zhou GD, Yi TH, Li HN (2014) Sensor placement optimization in structural health monitoring using cluster-in-cluster firefly algorithm. Adv Struct Eng 17(8):1103–1115
69. Zou D, Gao L, Li S, Wu J (2011) Solving 0-1 knapsack problem by a novel global harmony search algorithm. Appl Soft Comput 11:1556–1564