Knowledge-Based Systems 48 (2013) 17–23

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem

Ling Wang *, Xiao-long Zheng, Sheng-yao Wang

Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 7 December 2012
Received in revised form 7 April 2013
Accepted 7 April 2013
Available online 16 April 2013

Keywords:
Multidimensional knapsack problem
Fruit fly optimization algorithm
Binary algorithm
Smell-based search
Vision-based search

ABSTRACT

In this paper, a novel binary fruit fly optimization algorithm (bFOA) is proposed to solve the multidimensional knapsack problem (MKP). In the bFOA, binary string is used to represent the solution of the MKP, and three main search processes are designed to perform evolutionary search, including smell-based search process, local vision-based search process and global vision-based search process. In particular, a group generating probability vector is designed for producing new solutions. To enhance the exploration ability, a global vision mechanism based on differential information among fruit flies is proposed to update the probability vector. Meanwhile, two repair operators are employed to guarantee the feasibility of solutions. The influence of the parameter setting is investigated based on the Taguchi method of design of experiment. Extensive numerical testing results based on benchmark instances are provided. And the comparisons to the existing algorithms demonstrate the effectiveness of the proposed bFOA in solving the MKP, especially for the large-scale problems.

1. Introduction

The multidimensional knapsack problem (MKP) is a generalization of the standard 0–1 knapsack problem. The goal of the problem is to find a subset of all items that yields maximum profit without exceeding the multidimensional resource capacities. The MKP is a typical discrete optimization problem with wide applications. Many real problems can be formulated as the MKP, such as capital budgeting [1], resource allocating [2], cargo loading [3], cutting stock [4], knowledge and economics engineering [5], and pollution prevention and control [6]. The MKP has been shown to be strongly NP-hard [1].

In view of the pervasive nature of the MKP in knowledge-based system, there is abundant literature on the topic. Early solution algorithms are mainly exact methods, such as branch and bound algorithm (B&B) [3], dynamic programming (DP) [7], and the B&B and DP based hybridization method [8]. A recent breakthrough in developing effective methods for the MKP is the concept of core as described by Puchinger et al. [9]. Using the concept of core [10] can help to reduce the problem size significantly, which is useful in developing both the exact and approximate algorithms. Based on the concept of core, some effective algorithms have been developed for the knapsack problem (KP) [11,12], the MKP [9], the multiple-choice MKP [13], and the bi-objective MKP [14]. Besides, some heuristics have also been presented for the MKP [15–18]. With the development of computational intelligence, some metaheuristic methods have been proposed to solve the MKP. Relevant algorithms include simulated annealing [19], genetic algorithm [1], tabu search (TS) [20], linear programming with TS [21], ant colony optimization [22], max–min ant system [23], and particle swarm optimization [6]. However, most of the above methods cannot solve large-scale problems effectively and efficiently. Very recently, Wang et al. [24] proposed a hybrid algorithm based on estimation of distribution algorithm (HEDA) and designed a new repair operator particularly for large-scale problems. Due to the significance of the MKP in academic research and real applications, it is important to develop novel algorithms with satisfactory performances.

As a novel evolutionary optimization approach, fruit fly optimization algorithm (FOA) [25] is inspired by the knowledge from the foraging behavior of fruit flies. The FOA is easy to understand and implement. Some recent applications of the FOA can be found in literature, including financial distress [25], power load forecasting [26], web auction logistics service [27] and PID controller tuning [28,29]. As the traditional FOA was proposed to solve the continuous optimization problem, it operates on the continuous variables. For the discrete optimization problems, such as the MKP, the FOA should be specially designed to operate on the discrete variables, such as encoding scheme and search operators. To the best of our knowledge, there is no research work about the FOA for solving discrete optimization problems. To solve the MKP effectively, we shall propose a binary fruit fly optimization algorithm (bFOA) by considering the specific knowledge of the MKP, especially aiming

* Corresponding author. Tel.: +86 10 62783125; fax: +86 10 62786911.
E-mail address: wangling@mail.tsinghua.edu.cn (L. Wang).

0950-7051/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.04.003

at solving the large-scale problems. To be specific, solutions are represented as binary strings. The smell-based search process and the local vision-based search process of the traditional FOA are also used in the bFOA to perform exploitation based search. Different from the traditional FOA, a global vision-based search process is designed for the bFOA to enhance the exploration based search. Meanwhile, in the global vision-based search process, the population is generated according to a group generating probability vector, which is updated based on the differential information among fruit flies. In addition, two repair operators are employed to guarantee the feasibility of solutions. The influence of parameter setting is investigated based on the Taguchi method of design of experiment, and suitable values are recommended. Finally, extensive numerical testing results and comparisons are provided to demonstrate the effectiveness of the proposed bFOA.

The remainder of the paper is organized as follows. In Section 2, the mathematical formulation of the MKP is presented. In Section 3, the basic FOA is introduced. Then, the bFOA for the MKP is presented in details in Section 4. In Section 5, the influence of parameter setting is investigated, and numerical testing results and comparisons are given. Finally, we end the paper with some conclusions and future work in Section 6.

2. Multidimensional knapsack problem

Generally, the multidimensional knapsack problem can be formulated as follows [1]:

$$\text{Maximize } \sum_{j=1}^{n} c_j d_j \tag{1}$$

$$\text{subject to } \sum_{j=1}^{n} r_{i,j} d_j \leqslant s_i, \quad i = 1, 2, \ldots, m \tag{2}$$

$$d_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n \tag{3}$$

where n is the number of items, m is the number of knapsack constraints with capacity s_i (i = 1, 2, ..., m) at the ith dimension, c_j denotes the profit of item j, $r_{i,j}$ is the volume of item j at the ith dimension, and d_j is a binary decision variable. If item j is selected into the knapsack, then d_j = 1; otherwise, d_j = 0. Without loss of generality, it is assumed that c_j, $r_{i,j}$ and s_i are non-negative integers with $r_{i,j} \leqslant s_i$ and $\sum_{j=1}^{n} r_{i,j} > s_i,\ i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n$.

The objective of the MKP is to find a subset of all the items to be selected in the knapsack with a maximum profit while not exceeding the capacity of each dimension.

3. Fruit fly optimization algorithm

Fruit fly optimization algorithm (FOA) [25] is a new interactive evolutionary method inspired by the knowledge from foraging behavior of fruit flies. There are two main foraging processes: smell the food source by osphresis organ and fly towards the corresponding location; and use the sensitive vision to find food and fly towards a better direction. The foraging process of a fruit fly group can be illustrated in Fig. 1 [26], and the procedure of the FOA is summarized as follows [26].

Step 1. Initialize parameters, including maximum number of generations and population size.
Step 2. Initialize the fruit fly group.
Step 3. Use osphresis for foraging: generate several fruit flies randomly around the fruit fly group to construct a population:
Step 4. Evaluate the population to obtain the smell concentration values (fitness value) of each fruit fly.

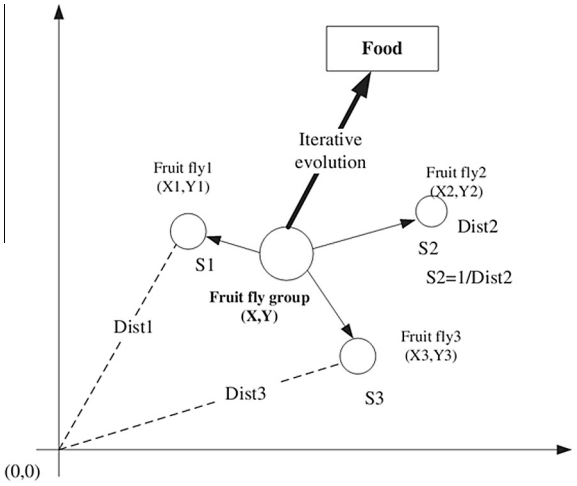

Fig. 1. Group iterative foraging process of fruit fly.

Step 5. Use vision for the foraging: find the best fruit fly with the maximum smell concentration value, and then let the fruit fly group fly towards the best one.
Step 6. End the algorithm if the maximum number of generations is reached; otherwise, go back to Step 3.

4. bFOA for MKP

In this section, we present the design of the bFOA for the MKP. Different from the traditional FOA, first we adopt a discrete binary string in the bFOA to represent a solution; second we use a group generating probability vector to generate population; third we adopt 0–1 flip operation to exploit the neighborhood in the smell-based search process; fourth we design a global vision-based search mechanism to enhance the exploration ability. Besides, two repair operations are adopted in the bFOA to guarantee the feasibility of solutions for further improving the performances of the bFOA.

4.1. Encoding scheme and population generating with probability vector

In the bFOA, each individual is a solution of the MKP, which is represented by an n-bit binary string. That is, for an individual x_j its ith bit $x_{i,j}$ (i = 1, 2, ..., n) is a binary decision variable, 0 or 1. Meanwhile, we used an n-dimensional vector $P(g) = [p_1(g), p_2(g), \ldots, p_n(g)]$, called group generating probability vector, to generate the fruit fly population. For an individual x_j, $p_i(g)$ indicates the probability of $x_{i,j} = 1$ at the gth generation. To uniformly sample the search space, the probability vector is initialized as $P(0) = [0.5, 0.5, \ldots, 0.5]$.

In each generation, a new population with N individuals is generated according to the probability vector. The procedure is shown in Fig. 2.

4.2. Smell-based search and local vision-based search

In the bFOA, S neighbors are randomly generated around each fruit fly x_j (j = 1, 2, ..., N) using the following smell-based search process.

Step 1. Randomly select L bits from x_j.
Step 2. Mutate the selected bits by replacing 0 with 1 or 1 with 0.

Procedure

1: **FOR** j=1 to N

2: **FOR** i=1 to n

3: Generate a uniformly random number $\zeta \in (0,1)$.

4: **IF** $\zeta < p_i(g)$ **THEN** Let $x_{i,j} = 1$.

5: **ELSE** Let $x_{i,j} = 0$.

7: **END IF**

8: **END FOR**

9: **END FOR**

Fig. 2. Generating population according to probability vector.

Considering a problem with 10 items, an example is illustrated in Fig. 3 for the above process, where L = 3, and the bold bits denote the selected ones.

Similar to the traditional FOA, each fruit fly in the bFOA also finds its best local neighbor using its sensitive vision. If the best neighbor is better, then it flies towards that neighbor. That is, the individual will be replaced with the best neighbor. Otherwise, it remains unchanged.

4.3. Global vision-based search

The traditional FOA focuses much on the exploitation ability. To enhance the exploration ability, we introduce a global vision-based mechanism in the bFOA to update the group generating probability vector. To be specific, inspired by the mechanism of differential evolution algorithm [30], the differential information among the best individual Fg and two randomly selected individuals $F1$, $F2$ are used to update $p_i(g)$.

$$\Delta_i = x_{i,Fg} + 0.5 \times (x_{i,F1} - x_{i,F2}) \quad (4)$$

$$p_i(g) = 1/[1 + e^{-b\times(\Delta_i - 0.5)}] \quad (5)$$

where Δ_i denotes the differential information, and b is a coefficient indicating the vision sensitivity.

After the group generating probability vector is updated, a new population is produced accordingly.

4.4. Repair operator

Usually, the newly generated individuals may not satisfy the capacity constraints. To guarantee the feasibility of the individuals, a repair procedure is often needed. In this paper, the following two repair operators are employed.

Repair operator 1 (RO1). The common used repair operator [1] is a greedy-like heuristic based on the pseudo-utility ratios, which ratios are usually determined using the surrogate duality approach [31]. Considering the surrogate relaxation problem of the MKP:

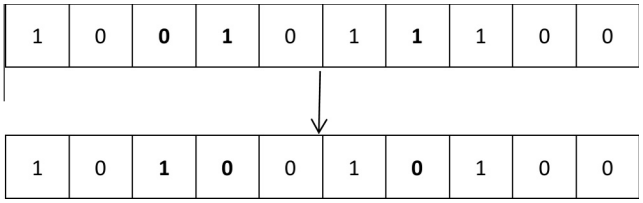

Fig. 3. An illustration of smell-based search.

Procedure

Let: I= {1,2,…,m}

1: Initialize $R_i = \sum_{j=1}^{n} r_{i,j} d_j$, $\forall\, i \in I$.

2: **FOR** j=n to 1 //DROP phase

3: **IF** (d_j =1) and ($\exists\, i \in I$, $R_i > s_i$) **THEN**

4: Let d_j =0, and $R_i = R_i - r_{i,j}$, $\forall\, i \in I$.

5: **END IF**

6: **END FOR**

7: **FOR** j=1 to n //ADD phase

8: **IF**($d_j = 0$) and ($\forall\, i \in I$, $R_i + r_{i,j} \le s_i$) **THEN**

9: Let d_j =1, and $R_i = R_i + r_{i,j}$, $\forall\, i \in I$.

10: **END IF**

11: **END FOR**

Fig. 4. Procedure of OR1.

Procedure

Let: I= {1,2,…,m}

1: Initialize $R_i = \sum_{j=1}^{n} r_{i,j} d_j$, $\forall\, i \in I$.

2: **FOR** i=1 to m //DROP phase

3: **FOR** j=n to 1

4: **IF** (d_j =1) and ($\exists\, i \in I$, $R_i > s_i$) **THEN**

5: Let d_j =0 and $R_i = R_i - r_{i,f_{i,j}}$, $\forall\, i \in I$.

6: **END IF**

7: **END FOR**

8: **END FOR**

9: Find the constraint i with the minimum of $s_i - R_i$.

10: **FOR** j=1 to n //ADD phase

11: **IF** ($d_j = 0$) and ($\forall\, i \in I$, $R_i + r_{i,f_{i,j}} \le s_i$) **THEN**

12: Let d_j =1 and $R_i = R_i + r_{i,f_{i,j}}$, $\forall\, i \in I$.

13: **END IF**

14: **END FOR**

Fig. 5. Procedure of OR2.

$$\text{Maximize} \sum_{j=1}^{n} c_j d_j \quad (6)$$

$$\text{subject to} \sum_{j=1}^{n} \left\{ \sum_{i=1}^{m} \omega_i r_{i,j} \right\} d_j \leqslant \sum_{i=1}^{m} \omega_i s_i \quad (7)$$

$$d_j \in \{0,1\}, \quad j = 1,2,\ldots,n \quad (8)$$

where ω_i is a positive real number as the surrogate multiplier (or weight).

When solving the LP relaxation of the original MKP, the values of the dual variables are the weights. That is, ω_i can be viewed as the shadow price of the ith constraint in the LP relaxation of the original MKP. Thus, the pseudo-utility ratio of each variable can be calculated as follows:

$$u_j = c_j \Big/ \sum_{i=1}^{m} \omega_i r_{i,j}, \quad j = 1, 2, \ldots, n \tag{9}$$

The core idea of this repair operator is to include or exclude each variable based on u_j. Before repairing, all the variables are sorted in a descent order according to the pseudo-utility ratios. Then, it performs repairing with two phases. The first phase (denoted DROP) is used to guarantee the feasibility of the solution. To be specific, it examines each bit of the encoded string in an ascent order of u_j and then changes the bit from 1 to 0 until the solution is feasible. The second phase (denoted ADD) is to improve the performance of the solution. To be specific, it changes the bit from 0 to 1 in a descent order of u_j as long as all the constraints are not violated. The pseudo-code of RO1 is illustrated in Fig. 4.

Repair operator 2 (RO2). When using RO1, it needs to solve the LP relaxation problem. Since it is difficult to solve the LP problem with large-scale, RO1 can be applied to small or medium-scale instances. Recently, Wang et al. [24] proposed a new repair operator considering the specific knowledge of the MKP for solving the large-scale MKP. With this repair operator, it is not necessary to solve the LP relaxation problem in advance. This operator is described as follows.

First, it obtains a matrix $Q = (q_{i,j} = c_j/r_{i,j})_{m\times n}$, where $q_{i,j}$ is the value of a unit volume (called value ratio) of the jth item in the ith constraint. Then, it uses an assistant matrix F to store the sorting rank of the item in a descent order of value ratio in all constraints. That is, $f_{i,j}$ represents the rank of the jth item in the ith constraint. Finally, we can design the corresponding DROP and ADD phases. In DROP phase, every constraint is checked by examining each bit and changing it from 1 to 0 in an ascending order of $q_{i,j}$ until the solution is feasible. In ADD phase, constraints are checked in an ascent order of the surplus capacity, and then it examines each bit and changes it from 0 to 1 in a descent order of $q_{i,j}$ as long as the constraint is not violated. The pseudo-code of this operator is illustrated in Fig. 5.

4.5. Procedure of bFOA

With the above specific design, the flowchart of the bFOA is illustrated in Fig. 6. It can be seen that the procedure consists of three main processes in addition to the initialization process. In smell-based search process, local neighbors are exploited. In local vision-based search process, the fruit flies are replaced by their best neighbors. In global vision-based search process, differential

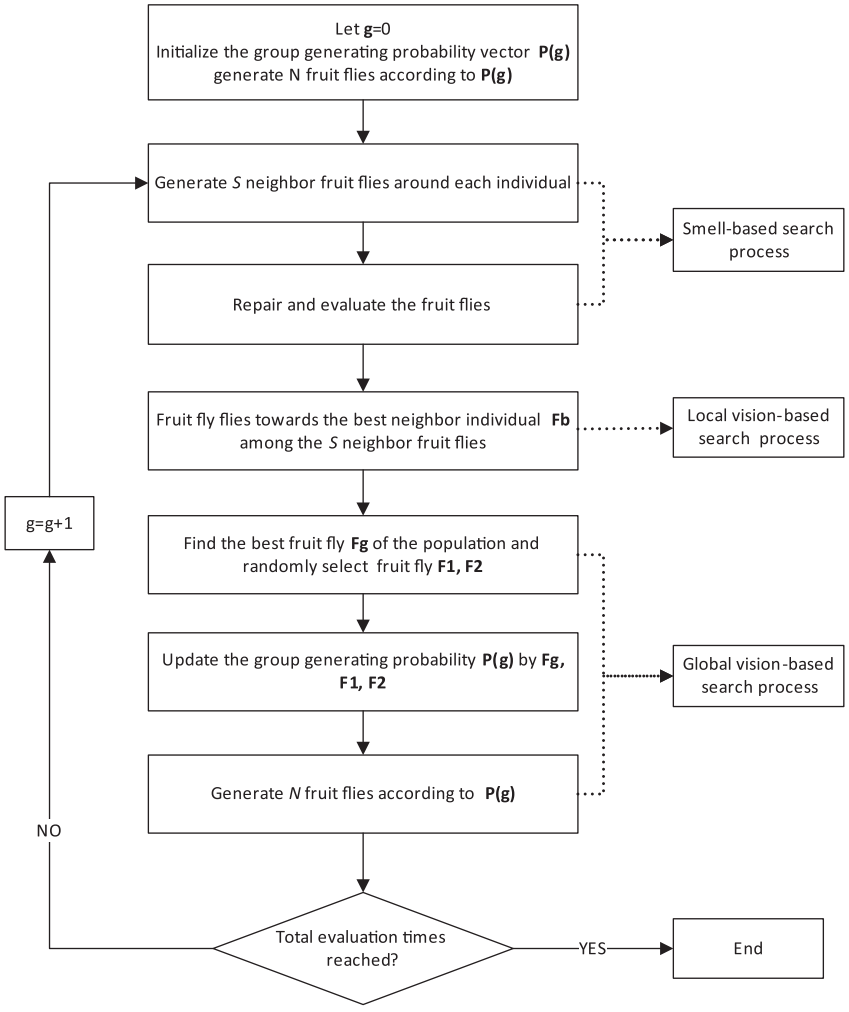

Fig. 6. Flowchart of bFOA.

Table 1
Combinations of parameter values.

Parameters	Factor level			
	1	2	3	4
N	20	30	40	50
S	1	3	5	10
L	1	3	5	7
b	20	30	40	50

Table 2
Orthogonal array and ARV value.

Experiment number	Factors				ARV
	N	*S*	*L*	*b*	
1	1	1	1	1	7637.18
2	1	2	2	2	7655.38
3	1	3	3	3	7651.40
4	1	4	4	4	7644.90
5	2	1	2	3	7654.60
6	2	2	1	4	7641.54
7	2	3	4	1	7648.76
8	2	4	3	2	7652.24
9	3	1	3	4	7652.08
10	3	2	4	3	7651.38
11	3	3	1	2	7643.24
12	3	4	2	1	7655.22
13	4	1	4	2	7649.84
14	4	2	3	1	7653.34
15	4	3	2	4	7655.50
16	4	4	1	3	7642.76

Table 3
Response value.

Level	*N*	*S*	*L*	*b*
1	7647.21	7648.42	7641.18	7648.63
2	7649.28	7650.41	7655.18	7650.18
3	7650.48	7649.72	7652.27	7650.03
4	7650.36	7648.78	7648.72	7648.50
Delta	3.27	1.99	14.00	1.67
Rank	2	3	1	4

information among the population is used to update the group generating probability vector for further generating new population to enhance exploration. Since exploitation and exploration are both considered in the algorithm, it expects to yield solutions with satisfactory quality.

In addition, we denote the algorithm as bFOA1 when RO1 is employed and as bFOA2 when RO2 is employed. Although we employ the repair operators from [24], we apply a totally different algorithm (bFOA) to solve the MKP. And this is the first work about the FOA to solve the MKP. In addition, the numerical results in the next section will show that our bFOA is better than the algorithm in [24], especially for solving the large-scale problems.

5. Numerical testing results and comparisons

To test the performance of the bFOA, two sets of well-known benchmark instances are used. Test set 1 consists of eight small-scale problems (available form ORLIB) with $m = 2–30$ and $n = 15–105$. Test set 2 consists of 11 medium-scale and large-scale problems (available from http://hces.bus.olemiss.edu/tools.html) with $m = 15–100$ and $n = 100–2500$. We code the algorithm in C++ and run it on a PC of Intel® Core2(TM) Q9550, 2.83 GHz.

5.1. Parameter setting

The proposed bFOA contains four key parameters: population size (N), the number of generated neighbors (S) and the number of the bits selected to flip (L) in the smell-based search process, and the global vision sensitivity (b). To investigate the influence of these parameters on the bFOA1, we apply the Taguchi method of design of experiment (DOE) [32] with instance Mk_gk06 ($m = 50$ and $n = 200$). Different combinations of the values are listed in Table 1.

With each combination of values, the bFOA1 is run 50 times independently (set the maximum evaluation times as 100,000). The average response variable (ARV) value is the average of total profit obtained by the bFOA1. Considering four factor levels for each parameter, we list the orthogonal array $L_{16}(4^4)$ and the obtained ARV values in Table 2. Using the statistical analysis tool Minitab, we can analyze the significance rank of each parameter

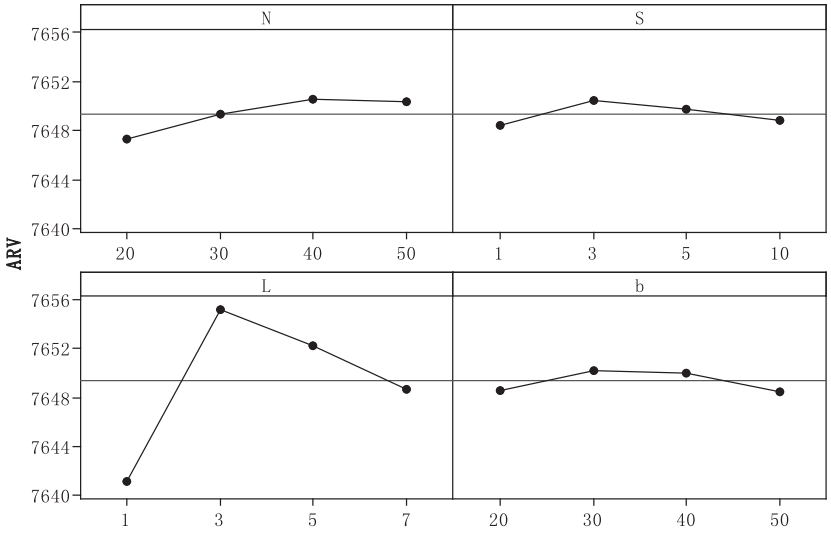

Fig. 7. Fact level trend of bFOA.

and obtain the trend of each factor level, as shown in Table 3 and Fig. 7, respectively.

From Table 3 and Fig. 7, it can be seen that L is the most significant parameter. A small value of L is helpful for local exploitation. But a too small value may be helpless for repair process. As for N, with a fixed total evaluation times, a small size may lead to poor exploration so as to cause premature convergence, while a large size implies a small number of evolution generations which may cause insufficient search. As for S and b, it can be seen from Fig. 7 that they have little influence on the algorithm, where medium values are preferable.

According to the DOE test based investigation, the suitable values of these parameters are set as $N = 40$, $S = 3$, $L = 3$ and $b = 30$, which will be used for the following testing.

5.2. Comparison of bFOA1 with HEDA1

Very recently, an algorithm named HEDA was proposed in [24] with better performances than other existing approaches. Therefore, we compare the bFOA1 with the HEDA1 (HEDA with RO1). Same as [24], we set the maximum number of evaluations as 100,000 and run the algorithm 20 times independently.

The comparative results with eight small-scale instances of test set 1 are listed in Table 4, where the column SR denotes the success ratio of hitting the best known value and AVG denotes the average value of the 20 runs. The comparative results with eight medium-scale instances of test set 2 are listed in Table 5, where Min.Dev and Ave.Dev represent the minimum and average percentage deviations from the best-known values, respectively, and Var. Dev denotes the variance of the deviation.

For the small-scale instances, it can be seen from Table 4 that both the bFOA1 and the HEDA1 can optimally solve the instances consistently. For the medium-scale instances, it can be seen from Table 5 that the bFOA1 is superior to the HEDA1 in terms of Min.Dev, Ave.Dev and Var.Dev. It implies that the bFOA1 is able to obtain better solutions than the HEDA1 in a more consistent way.

Table 4
Comparisons of bFOA1 with HEDA1 on small scale MKP.

Problem	$n \times m$	Best known	bFOA1		HEDA1	
			AVG	SR	AVG	SR
Sent01	60 × 30	7772	7772	100	7772	100
Sent02	60 × 30	8722	8722	100	8722	100
Weing7	105 × 2	1,095,445	1,095,445	100	1,095,445	100
Weing8	105 × 2	624,319	624,319	100	624,319	100
Knap15	15 × 10	4015	4015	100	4015	100
Knap20	20 × 10	6120	6120	100	6120	100
Knap39	39 × 5	10,618	10,618	100	10,618	100
Knap28	28 × 10	12,400	12,400	100	12,400	100

5.3. Comparison of bFOA2 with HEDA2

In [24], RO2 was incorporated in the HEDA as the HEDA2 for solving the large-scale instances. Similarly, we incorporate RO2 in the bFOA as the bFOA2. Then, we compare the bFOA2 with the HEDA2 using all the instances of test set 2. The comparative results are listed in Table 6.

From Table 6, it is clear that the bFOA2 dominates the HEDA2 for all the instances. So, our bFOA2 is more powerful than the HEDA2 to solve the MKP, including the large-scale instance with 2500 items.

In addition, comparing the results of the bFOA1 in Table 5 with those of the bFOA2 in Table 6, it can be seen that RO1 is superior to RO2 for the algorithm to obtain better solutions. But, RO1 is difficult to be applied for the large-scale problems. Thus, the bFOA1 is recommended for the small and medium-scale problems, while the bFOA2 is recommended for the large-scale problems.

Table 5
Comparisons of bFOA1 with HEDA1 on medium scale MKP.

Problem	$n \times m$	Best known	bFOA1			HEDA1		
			Min.Dev	Ave.Dev	Var.Dev	Min.Dev	Ave.Dev	Var.Dev
Mk_gk01	100 × 25	3766	**0.2390**	**0.3053**	**0.0332**	0.2655	0.3173	0.0783
Mk_gk02	100 × 50	3958	**0**	**0.1983**	0.2306	0.0758	**0.1983**	**0.1194**
Mk_gk03	150 × 25	5650	**0.0708**	**0.1584**	**0.0946**	0.1239	0.2018	0.0998
Mk_gk04	150 × 50	5764	**0.0520**	**0.1943**	0.3567	0.0867	0.3348	**0.2916**
Mk_gk05	200 × 25	7557	**0.0529**	**0.1198**	**0.0787**	0.1323	0.2772	0.2150
Mk_gk06	200 × 50	7672	**0.1173**	**0.2046**	**0.1214**	0.2998	0.4438	0.2339
Mk_gk07	500 × 25	19,215	**0.0520**	**0.0822**	**0.0472**	0.5204	0.7203	0.8240
Mk_gk08	500 × 50	18,801	**0.0851**	**0.1402**	**0.1267**	0.6117	0.7680	1.1028

Note: The bold values denote better results.

Table 6
Comparisons of bFOA2 with HEDA2.

Problem	$n \times m$	Best known	bFOA2			HEDA2		
			Min.Dev	Ave.Dev	Var.Dev	Min.Dev	Ave.Dev	Var.Dev
Mk_gk01	100 × 25	3766	**0.5576**	**0.7554**	**0.3305**	0.6107	0.9360	0.6555
Mk_gk02	100 × 50	3958	**0.6569**	**0.8518**	**0.3317**	0.7580	0.9790	0.4039
Mk_gk03	150 × 25	5650	**0.7965**	**0.9150**	**0.4126**	0.9381	1.1531	0.7828
Mk_gk04	150 × 50	5764	**0.8675**	**1.0279**	**0.4804**	1.0930	1.2673	0.7069
Mk_gk05	200 × 25	7557	**1.0057**	**1.1930**	**0.5905**	1.2439	1.4960	0.7483
Mk_gk06	200 × 50	7672	**0.8472**	**0.9802**	**0.4244**	1.1210	1.4436	1.1573
Mk_gk07	500 × 25	19,215	**1.4312**	**1.5194**	**0.5784**	1.7018	1.8460	0.9873
Mk_gk08	500 × 50	18,801	**1.2872**	**1.3659**	0.3551	1.6595	1.7260	**0.2880**
Mk_gk09	1500 × 25	58,085	**2.1744**	**2.2816**	0.9267	2.4585	2.5461	**0.6813**
Mk_gk10	1500 × 50	57,292	**1.7437**	**1.7905**	**0.4148**	2.0177	2.1170	0.7807
Mk_gk11	2500 × 100	95,231	**1.5037**	**1.5738**	0.5780	1.7043	1.7348	**0.2184**

Note: The bold values denote better results.

6. Conclusions

This was the first reported work of using fruit fly optimization to solve the multidimensional knapsack problem. We designed a binary fruit fly optimization algorithm with three main search processes, including the smell-based search, local vision-based search and global vision-based search. In particular, a group generating probability vector was designed to generate population and a mechanism for the global vision-based search was provided to update the probability vector for well exploration. Besides, two repair operators were employed to guarantee the feasibility of solutions. The influence of parameter setting was investigated and suitable values were suggested. Numerical testing results and comparisons demonstrated the effectiveness of the proposed bFOA. To further improve the performance of the bFOA for solving the MKP, it is worthy of applying the concept of core in designing the algorithm. Besides, our future work includes the design of the FOA-based approaches for solving other combinatorial optimization problems like shop scheduling, and it is also important to study the FOA for solving the multi-objective optimization problems.

Acknowledgments

The authors thank the Editor-in-Chief Prof. Hamido Fujita and the anonymous reviewers for their constructive comments to improve the paper. This research is partially supported by the National Key Basic Research and Development Program of China (No. 2013CB329503), the National Science Foundation of China (No. 61174189), and the National Science and Technology Major Project of China (No. 2011ZX02504-008).

References

[1] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1998) 63–86.
[2] B. Gavish, H. Pirkul, Allocation of databases and processors in a distributed computing system, Management of Distributed Data Processing (1982) 215–231.
[3] W. Shih, A branch and bound method for the multi-constraint zero–one knapsack problem, Journal of the Operational Research Society 30 (1979) 369–378.
[4] P.C. Gilmore, R.E. Gomory, The theory and computation of knapsack functions, Operations Research 14 (1966) 1045–1075.
[5] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.
[6] J.C. Bansal, K. Deep, A modified binary particle swarm optimization for knapsack problems, Applied Mathematics and Computation 218 (2012) 11042–11061.
[7] P. Toth, Dynamic programing algorithms for the zero–one knapsack problem, Computing 25 (1980) 29–45.
[8] G. Plateau, M. Elkihel, A hybrid method for the 0–1 knapsack problem, Methods of Operations Research 49 (1985) 277–293.
[9] J. Puchinger, G.R. Raidl, U. Pferschy, The core concept for the multidimensional knapsack problem, Lecture Notes on Computer Science 3906 (2006) 195–208.
[10] E. Balas, E. Zemel, An algorithm for large zero–one knapsack problems, Operations Research 28 (1980) 1130–1154.
[11] S. Martello, P. Toth, A new algorithm for the 0–1 knapsack problem, Management Science 34 (1988) 633–644.
[12] D. Pisinger, An expanding-core algorithm for the exact 0–1 knapsack problem, European Journal of Operational Research 87 (1995) 175–187.
[13] M.R. Razzazi, T. Ghasemi, An exact algorithm for the multiple-choice multidimensional knapsack based on the core, Communications in Computer and Information Science 6 (2009) 275–282.
[14] G. Mavrotas, J.R. Figueira, A. Antoniadis, Using the idea of expanded core for the exact solution of bi-objective multi-dimensional knapsack problems, Journal of Global Optimization 49 (2011) 589–606.
[15] R. Loulou, E. Michaelides, New greedy-like heuristics for the multidimensional 0–1 knapsack problem, Operational Research 27 (1979) 1101–1114.
[16] Y. Toyoda, A simplified algorithm for obtaining approximate solutions to zero–one programming problems, Management Science 21 (1975) 1417–1427.
[17] J.E. Beasley, Multidimensional knapsack problems, in: C.A. Floudas, P.M. Pardalos (Eds.), Encyclopedia of Optimization, 2nd ed., Springer-Verlag, US, 2009, pp. 2402–2406.
[18] F.D. Croce, A. Grosso, Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem, Computers & Operations Research 39 (2012) 27–31.
[19] A. Drexl, A simulated annealing approach to the multi-constraint zero–one knapsack problem, Computing 40 (1988) 1–8.
[20] F. Dammeyer, S. Voss, Application of tabu search strategies for solving multi-constraint zero–one knapsack problems, Working paper, 1991.
[21] M. Vasquez, Y. Vimont, Improved results on the 0–1 multidimensional knapsack problem, European Journal of Operational Research 165 (2005) 70–81.
[22] M. Kong, P. Tian, Y.C. Kao, A new ant colony optimization algorithm for the multidimensional knapsack problem, Computers & Operations Research 35 (2008) 2672–2683.
[23] L. Ke, Z. Feng, Z. Ren, X. Wei, An ant colony optimization approach for the multidimensional knapsack problem, Journal of Heuristics 16 (2010) 65–83.
[24] L. Wang, S. Wang, Y. Xu, An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem, Expert Systems with Applications 39 (2012) 5593–5599.
[25] W.T. Pan, A new fruit fly optimization algorithm: taking the financial distress model as an example, Knowledge-Based Systems 26 (2012) 69–74.
[26] H. Li, S. Guo, C. Li, J. Sun, A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm, Knowledge-Based Systems 37 (2013) 378–387.
[27] S.M. Lin, Analysis of service satisfaction in web auction logistics service using a combination of fruit fly optimization algorithm and general regression neural network, Neural Computing & Applications (2011), http://dx.doi.org/10.1007/s00521-011-0769-1.
[28] J. Han, P. Wang, X. Yang, Tuning of PID controller based on fruit fly optimization algorithm, International Conference on Mechatronics and Automation (ICMA) (2012) 409–413.
[29] C. Li, S. Xu, W. Li, L. Hu, A novel modified fly optimization algorithm for designing the self-tuning proportional integral derivative controller, Journal of Convergence Information Technology 7 (2012) 69–77.
[30] R. Storn, K. Price, Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[31] H. Pirkul, A heuristic solution procedure for the multiconstraint zero–one knapsack problem, Naval Research Logistics 34 (1987) 161–172.
[32] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, Arizona, 2005.