

# Improved Binary Artificial Fish Swarm Algorithm for the 0–1 Multidimensional Knapsack Problems

Md. Abul Kalam Azad<sup>a,\*</sup>, Ana Maria A.C. Rocha<sup>a,b</sup>, Edite M.G.P. Fernandes<sup>a</sup>

<sup>a</sup>*Algoritmi R&D Centre*

<sup>b</sup>*Department of Production and Systems*

*School of Engineering, University of Minho, 4710-057 Braga, Portugal*

---

## Abstract

The 0–1 multidimensional knapsack problem (MKP) arises in many fields of optimization and is NP-hard. Several exact as well as heuristic methods exist. Recently, an artificial fish swarm algorithm has been developed in continuous global optimization. The algorithm uses a population of points in space to represent the position of fish in the school. In this paper, a binary version of the artificial fish swarm algorithm is proposed for solving the 0–1 MKP. In the proposed method, a point is represented by a binary string of 0/1 bits. Each bit of a trial point is generated by copying the corresponding bit from the current point or from some other specified point, with equal probability. Occasionally, some randomly chosen bits of a selected point are changed from 0 to 1, or 1 to 0, with an user defined probability. The infeasible solutions are made feasible by a decoding algorithm. A simple heuristic `add_item` is implemented to each feasible point aiming to improve the quality of that solution. A periodic reinitialization of the population greatly improves the quality of the solutions obtained by the algorithm. The proposed method is tested on a set of benchmark instances and a comparison with other methods available in literature is shown. The comparison shows that the proposed method gives a competitive performance when solving this kind of problems.

*Keywords:* 0–1 knapsack problem, multidimensional knapsack, artificial fish swarm, decoding algorithm

---

---

\*Corresponding author

*Email addresses:* `akazad@dps.uminho.pt` (Md. Abul Kalam Azad), `arocha@dps.uminho.pt` (Ana Maria A.C. Rocha), `emgpf@dps.uminho.pt` (Edite M.G.P. Fernandes)

1 **1. Introduction**

2 The 0–1 multidimensional knapsack problem (MKP) is a NP-hard combinatorial  
 3 optimization problem that arises in many practical problems, such as capital bud-  
 4 geting and project selection problem [1, 2], allocating processors and databases in a  
 5 distributed computer system [3], project selection, cargo loading and so on [4]. The  
 6 0–1 MKP is formulated as follows:

$$\begin{aligned} & \text{maximize } z(\mathbf{x}) \equiv \mathbf{c}\mathbf{x} \\ & \text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \quad x_j \in \{0, 1\}, \quad j = 1, 2, \dots, n, \end{aligned} \tag{1}$$

7 where  $\mathbf{c} = (c_1, c_2, \dots, c_n)$  is an  $n$ -dimensional row vector of profits,  $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$   
 8 is an  $n$ -dimensional column vector of 0–1 decision variables,  $\mathbf{A} = [a_{k,j}]$ ,  $k = 1, 2, \dots, m$ ,  
 9  $j = 1, 2, \dots, n$  is an  $m \times n$  coefficient matrix of resources and  $\mathbf{b} = (b_1, b_2, \dots, b_m)^T$  is  
 10 an  $m$ -dimensional column vector of resource capacities. It should be noted here that,  
 11 in a 0–1 multidimensional knapsack problem, each element of  $\mathbf{c}$ ,  $\mathbf{A}$  and  $\mathbf{b}$  is assumed  
 12 to be nonnegative. The goal of the 0–1 MKP is to find a subset of  $n$  items that  
 13 yields maximum profit  $z$  without exceeding resource capacities  $\mathbf{b}$ . There are many  
 14 knapsack-like problems. The knapsack family includes the 0–1 knapsack problem  
 15 (KP), where there is just one single constraint ( $m = 1$ ). Effective approximate al-  
 16 gorithms have been developed for obtaining its near optimal solutions. If a bounded  
 17 amount of each item type is available, then the bounded knapsack problem arises. On  
 18 the other hand, the unbounded knapsack problem is a generalization of the bounded  
 19 knapsack problem where an unlimited number of each item type is available. Another  
 20 generalization of the 0–1 knapsack problem is the multiple-choice knapsack problem  
 21 where exactly one item from each of several classes of items is chosen such that the  
 22 profit  $z$  is to be maximized. If the profits equal the resources  $c_j = a_j$ ,  $j = 1, 2, \dots, n$ ,  
 23 then one has the subset-sum problem. The name comes from the fact that it can  
 24 be seen as the problem of choosing a subset of  $c_1, c_2, \dots, c_n$  such that its sum is as  
 25 large as possible without exceeding  $b$ . The multiple knapsack problem appears when  
 26 some of the  $n$  items are chosen to pack in  $m$  knapsacks of (maybe) different capac-  
 27 ities such that the profit  $z$  is maximized. The most general form of the knapsack  
 28 problem is the multidimensional knapsack problem where all coefficients  $c_j, a_{k,j}, b_k$ ,  
 29  $k = 1, 2, \dots, m, j = 1, 2, \dots, n$  and variables  $x_j, j = 1, 2, \dots, n$  are nonnegative inte-  
 30 gers, which turns out to be a general integer programming problem [5].

31 In the last decades several exact as well as heuristic methods have been pro-  
 32 posed to solve the MKP. Exact methods include dynamic programming methods  
 33 [6, 7], branch-and-bound algorithms [3, 4, 8], the Fourier-Motzkin elimination based  
 34 enumeration algorithms [9], asymptotic analysis method [10], statistical analysis

1 method [11], linked LP-relaxations, disjunctive cuts and implicit enumeration [12],  
2 generalized fuzzy approach [13], core concept based on LP-relaxation [14] and so on.  
3 Pisinger [5] has proposed several exact algorithms for solving knapsack problems in  
4 his doctoral thesis. A list of some heuristic methods for solving the MKP follows.  
5 Drexler [15] proposed a simulated annealing based on the add-interchange-drop tech-  
6 nique for handling the constraints. Hanafi and Fréville [16] proposed a tabu search  
7 approach for the 0–1 MKP using the surrogate constraints information. Vasquez and  
8 Vimont in [17] presented a hybrid method that combines the linear programming with  
9 an efficient tabu search. Chu and Beasley [18] proposed the most successful genetic  
10 algorithm (GA) for solving the 0–1 MKP. The authors present the drop-add repair  
11 operator based on the pseudo-utility ratios in order to make the solutions feasible.  
12 Sakawa and Kato [19] introduced a genetic algorithm with double strings (GADS)  
13 based on a decoding algorithm. In this decoding algorithm, the items that make  
14 the solution infeasible are dropped from the solution by checking all the constraints.  
15 Djannaty and Doosdar investigated in [20] a hybrid genetic algorithm that uses a  
16 penalty function. A binary ant colony optimization algorithm based on the drop-add  
17 repair operator [18] is provided in [21]. Zou et al. [22] have recently developed a novel  
18 global harmony search algorithm based on a penalty function for a KP. Some other  
19 heuristics are available in the literature [23, 24, 25, 26, 27]. An interesting review of  
20 different solution methods for solving the 0–1 MKP is found in [28]. The focus of  
21 the paper is on the theoretical properties and contains an overview of approximate  
22 and exact solution methods.

23 The artificial fish swarm algorithm (AFSA) that simulates the behavior of a fish  
24 school inside water was recently designed and applied in an engineering context  
25 [29, 30, 31, 32]. Fishes desire to stay close to the school to protect themselves from  
26 predators and to look for food, and to avoid collisions within the group. The main  
27 fish school behavior are the following: random, chasing, swarming, searching and  
28 leaping.

29 The artificial fish is a fictitious entity of a true fish. When applied to an opti-  
30 mization problem, a ‘fish’ within the school is represented by a point, also known  
31 as a candidate solution, and the school is the so-called population, or set of points  
32 or solutions. Inspired by fish school behavior, researchers have developed numeri-  
33 cal algorithms aiming to converge to a global optimal solution of the optimization  
34 problem, in an efficient manner. The environment in which the artificial fish moves,  
35 searching for the optimum, is the feasible search space of the problem.

36 A novel fish swarm heuristic which gives priority to the chasing behavior in detri-  
37 ment of the swarming one, for box constrained global optimization problems, was  
38 recently presented in [33]. Rocha et al. [34] developed an augmented Lagrangian fish

1 swarm based method for globally solving a nonlinear general constrained problem.  
2 A state-of-the-art regarding hybridizations and applications of the AFSA has just  
3 appeared in [35].

4 Binary versions of the most popular stochastic algorithms are common for solving  
5 discrete binary optimization problems [36, 37], namely 0–1 MKP [38, 39, 40, 41, 42].  
6 Based on AFSA for continuous global optimization, in this paper, we propose an  
7 improved binary version of the artificial fish swarm algorithm (IbAFSA) for solving  
8 the 0–1 MKP (1). A preliminary binary version of the artificial fish swarm algorithm  
9 (bAFSA) has been presented in [43]. The algorithm was tested on a small set of  
10 problems. For the sake of simplicity, while describing the proposed binary AFSA  
11 we will use the words ‘point’ to represent the position of a fish in the school, and  
12 ‘population’ to denote the fish school.

13 In the present study, all points in the population are randomly initialized, each  
14 represented by a binary 0/1 string of length  $n$ . The procedure that checks which  
15 points are in the vicinity of each individual point, the so-called ‘visual scope’, is  
16 carried out using the Hamming distance. When chasing, searching or swarming  
17 behavior are selected, the proposed IbAFSA generates each bit of the trial point by  
18 copying the corresponding bit from the current point or from a second point, with  
19 equal probability. In chasing, the second point is the best point inside the ‘visual  
20 scope’, and in searching, that point is randomly selected from the ‘visual scope’. For  
21 the swarming behavior, the second point is the central point that is computed based  
22 on ideas presented in [44]. We remark that in the previous work [43], when swarming  
23 was implemented, a bit of the current point was randomly selected and changed from  
24 0 to 1 or vice versa to create the trial point. Furthermore, the infeasible solutions are  
25 made feasible using an adapted version of the decoding algorithm presented in [19].  
26 Along with the decoding algorithm, an `add_item` operation is also implemented to  
27 each feasible solution aiming to increase the profit throughout the addition of more  
28 items in the knapsack. To improve the quality of the solutions obtained by the  
29 algorithm, the population is periodically reinitialized.

30 Thus, the novel contributions of the presented IbAFSA, when compared with  
31 bAFSA [43], are: i) the computation of a central point inside the ‘visual scope’  
32 to define a point that is closest to all the other points in the ‘visual scope’, for the  
33 swarming behavior; ii) the implementation of a different strategy to generate the trial  
34 point using the current and the central point, in swarming; iii) the implementation of  
35 an `add_item` operation to each feasible point; iv) the reinitialization of the population  
36 periodically, although keeping the best point of the population. The performance  
37 of the proposed IbAFSA is tested on a benchmark set of 0–1 MKP test instances.  
38 Although the proposal is very simple and easy-to-implement, the comparisons carried

1 out until now show that the algorithm is a competitive alternative to other heuristic  
 2 methods from the literature.

3 A crucial motivation to assess the performance of IbAFSA on the 0–1 MKP is  
 4 that several test problem instances together with their known optimal solution are  
 5 available in the literature.

6 The organization of this paper is as follows. We briefly describe the artificial fish  
 7 swarm algorithm in Section 2. In Section 3 the proposed improved binary artificial  
 8 fish swarm algorithm is outlined. Section 4 describes the experimental results and  
 9 finally we draw the conclusions of this study in Section 5.

## 10 2. Artificial Fish Swarm Algorithm

11 In this section, we give a brief description of AFSA proposed in [33] for box  
 12 constrained global optimization problems of type minimize  $\mathbf{x} \in \Omega f(\mathbf{x})$ . Here  $f : \mathbb{R}^n \rightarrow$   
 13  $\mathbb{R}$  is a nonlinear function that is to be minimized and  $\Omega = \{\mathbf{x} \in \mathbb{R}^n : l_j \leq x_j \leq$   
 14  $u_j, j = 1, 2, \dots, n\}$  is the search space.  $l_j$  and  $u_j$  are the lower and upper bounds of  
 15  $x_j$ , respectively, and  $n$  is the number of variables of the optimization problem.

AFSA works with a population of  $N$  points  $\mathbf{x}^i, i = 1, 2, \dots, N$  to identify promis-  
 ing regions looking for a global solution [31].  $\mathbf{x}^i$  is a floating-point encoding that  
 covers the entire search space  $\Omega$ . The crucial issue of AFSA is the ‘visual scope’ of  
 each point  $\mathbf{x}^i$ . This represents a closed neighborhood of  $\mathbf{x}^i$  with a radius equal to a  
 positive quantity  $\nu$  defined by

$$\nu = \delta \max_{j \in \{1, 2, \dots, n\}} (u_j - l_j)$$

16 where  $\delta \in (0, 1)$  is a positive visual parameter. This parameter may be reduced  
 17 along the iterative process. Let  $\mathbf{I}^i$  be the set of indices of the points inside the  
 18 ‘visual scope’ of point  $\mathbf{x}^i$ , where  $i \notin \mathbf{I}^i$  and  $\mathbf{I}^i \subset \{1, 2, \dots, N\}$ , and let  $np^i$  be the  
 19 number of points in its ‘visual scope’. Depending on the relative positions of the  
 20 points in the population, three possible situations may occur:

- 21 a) when  $np^i = 0$ , the ‘visual scope’ is empty, and the point  $\mathbf{x}^i$ , with no other  
 22 points in its neighborhood, moves randomly looking for a better region;
- 23 b) when the ‘visual scope’ is not crowded, the point  $\mathbf{x}^i$  is able either to chase  
 24 moving towards the best point inside the ‘visual scope’, or, if this best point  
 25 does not improve the objective function value corresponding to  $\mathbf{x}^i$ , to swarm  
 26 moving towards the central point of the ‘visual scope’;

- 1 c) when the ‘visual scope’ is crowded, the point  $\mathbf{x}^i$  has some difficulty in following  
 2 any particular point, and searches for a better region by choosing randomly  
 3 another point (from the ‘visual scope’) and moving towards it.
- 4 The condition that decides when the ‘visual scope’ of  $\mathbf{x}^i$  is not crowded is

$$C_f \equiv \frac{np^i}{N} \leq \theta, \quad (2)$$

where  $C_f$  is the crowding factor and  $\theta \in (0, 1)$  is the crowd parameter. In this situation, the point  $\mathbf{x}^i$  has the ability to swarm or to chase. The swarming behavior is characterized by a movement towards the central point inside the ‘visual scope’ of  $\mathbf{x}^i$  defined by

$$\bar{\mathbf{x}} = \frac{\sum_{l \in \mathbf{I}^i} \mathbf{x}^l}{np^i}.$$

- 5 We refer the reader to [31, 32, 33, 34] for details.

### 6 **3. Improved Binary Artificial Fish Swarm Algorithm**

7 In this section we will present the proposed IbaAFSA to solve the 0–1 multidimensional knapsack problem (1). The outline of the algorithm is described in the  
 8 following.  
 9

#### 10 *3.1. Initialization (coding)*

11 The first step to design the IbaAFSA for solving the 0–1 MKP is to devise a  
 12 suitable representation scheme of a point/solution from the population. Since we  
 13 consider the 0–1 knapsack problem,  $N$  solutions,  $\mathbf{x}^i, i = 1, 2, \dots, N$  are randomly  
 14 initialized, each represented by a binary 0/1 string of length  $n$  [43, 45, 46]. We  
 15 remark that the maximum population size  $N$  of binary 0/1 strings of length  $n$  is  $2^n$ .

#### 16 *3.2. Generating trial points in IbaAFSA*

17 In IbaAFSA the Hamming distance,  $H_d$ , is used to identify the points inside the  
 18 ‘visual scope’ of point  $\mathbf{x}^i$ . The Hamming distance between two bit sequences of equal  
 19 length is the number of positions at which the corresponding bits are different. After  
 20 calculating the Hamming distance between all pair of points from the population, the  
 21  $np^i$  points inside the ‘visual scope’ of  $\mathbf{x}^i$  are identified as the points  $\mathbf{x}^j$  that satisfy  
 22 the condition  $H_d(\mathbf{x}^i, \mathbf{x}^j) \leq \nu$ , for  $j \in \{1, \dots, N\}, j \neq i$ , where

$$\nu = \delta \times n, \quad (3)$$

1  $\delta \in (0, 1)$  and  $n$  represents the maximum Hamming distance between two binary  
 2 points. After computing  $np^i$ , the crowding factor  $C_f$  of  $\mathbf{x}^i$  is calculated using (2).

3 Depending on the value of  $C_f$ , the ‘visual scope’ can be empty, not crowded or  
 4 crowded. In IbAFSA, the behavior that generate the trial points are outlined as  
 5 follows.

### 6 3.2.1. Chasing behavior

7 If the ‘visual scope’ of  $\mathbf{x}^i$  is not crowded and the point that has the best objec-  
 8 tive function value inside the ‘visual scope’, denoted by  $\mathbf{x}^{\text{best}}$  ( $\text{best} \in \mathbf{I}^i$ ), satisfies  
 9  $z(\mathbf{x}^{\text{best}}) > z(\mathbf{x}^i)$ , the chasing behavior is to be implemented. In chasing, each bit of  
 10 the trial point,  $\mathbf{y}^i$ , is generated by copying the corresponding bit from  $\mathbf{x}^i$  or from  $\mathbf{x}^{\text{best}}$   
 11 with equal probability. This operation is similar to the uniform crossover present in  
 12 genetic/evolutionary algorithms.

### 13 3.2.2. Swarming behavior

14 When the ‘visual scope’ is not crowded and  $z(\mathbf{x}^{\text{best}}) \leq z(\mathbf{x}^i)$  (chasing is not  
 15 possible), then if  $z(\bar{\mathbf{x}}) > z(\mathbf{x}^i)$ , where  $\bar{\mathbf{x}}$  is the central point inside the ‘visual scope’  
 16 of the point  $\mathbf{x}^i$ , the swarming behavior is to be implemented. The central  $\bar{\mathbf{x}}$  is  
 17 the point closest to all the other points in the ‘visual scope’, in the sense that the  
 18 average Hamming distance to all other points in the ‘visual scope’ is minimal. Since  
 19 in IbAFSA, the points are represented by binary 0/1 strings, each bit of  $\bar{\mathbf{x}}$  takes  
 20 the majority of the corresponding bits of the other points in the ‘visual scope’, and  
 21 is randomly defined in case of tie. We refer to [44] for details. The pseudocode to  
 compute the central point is shown in Algorithm 1. In swarming, each bit of the

---

#### Algorithm 1 Central point

---

**Require:** Set  $\mathbf{I}^i$  and the  $np^i$  points inside the ‘visual scope’ of  $\mathbf{x}^i$

```

1: for  $j = 1$  to  $n$  do
2:   Compute  $\bar{x}_j = \frac{\sum_{l \in \mathbf{I}^i} x_j^l}{np^i}$ 
3:   if  $\bar{x}_j = 0.5$  then
4:     Set  $\bar{x}_j := \text{Random Integer}\{0, 1\}$ 
5:   else
6:     Set  $\bar{x}_j := \text{Round}(\bar{x}_j)$ 
7:   end if
8: end for
9: return Central point  $\bar{\mathbf{x}}$ 

```

---

22 trial  $\mathbf{y}^i$  is created by copying the corresponding bit from  $\mathbf{x}^i$  or from  $\bar{\mathbf{x}}$  with equal  
 23 probability.  
 24

1 *3.2.3. Searching behavior*

2 The searching behavior is tried in the following situations:

- 3 a) when the ‘visual scope’ is not crowded and neither  $\mathbf{x}^{\text{best}}$  nor  $\bar{\mathbf{x}}$  improves in  
4 objective function value;
- 5 b) when the ‘visual scope’ is crowded.

6 Here, a point  $\mathbf{x}^{\text{rand}}$  ( $\text{rand} \in \mathbf{I}^i$ ) inside the ‘visual scope’ of  $\mathbf{x}^i$  is randomly selected  
7 and the searching behavior is to be implemented if  $z(\mathbf{x}^{\text{rand}}) > z(\mathbf{x}^i)$ . Otherwise, a  
8 random behavior is implemented. In searching, each bit of  $\mathbf{y}^i$  is created by copying  
9 the corresponding bit from  $\mathbf{x}^i$  or  $\mathbf{x}^{\text{rand}}$  with equal probability.

10 *3.2.4. Random behavior*

11 When the ‘visual scope’ of  $\mathbf{x}^i$  is empty or the other behavior were not performed,  
12 the point  $\mathbf{x}^i$  performs the random behavior. In this case, the trial point  $\mathbf{y}^i$  is created  
13 by randomly setting a binary string of 0/1 bits of length  $n$ .

14 *3.3. Constraints handling*

15 The widely used approach to deal with constraints is based on penalty functions  
16 where a penalty term is added to the objective function in order to penalize the  
17 constraint violation. The penalty function method can be applied to any type of  
18 constraints, but the performance of penalty-type method is not always satisfactory  
19 due to the choice of an appropriate penalty parameter. Although several ideas have  
20 been proposed about designing efficient penalty functions and tuning penalty param-  
21 eters [20, 22], other alternative constraint handling techniques have emerged in the  
22 last decades.

23 There are a number of standard ways of dealing with constraints and infeasible  
24 solutions in binary represented population-based methods. In IbAFSA, the decoding  
25 algorithm proposed by Sakawa and Kato [19] to make infeasible solutions feasible is  
26 used. Although GADS and IbAFSA use different point representations, we modify  
27 the decoding algorithm so that it can decode points in a population in the same way  
28 as in [19]. The advantage of this algorithm is that decoding a point  $\mathbf{x}^i$  starts from any  
29 index and randomly continues to select an index until the maximum length of string  
30  $n$  is reached to make the point  $\mathbf{x}^i$  feasible, aiming to obtain promising solution (and  
31 hopefully optimal). At first, a set  $\mathbf{J} = \{J_1, J_2, \dots, J_n\}$  is defined with  $n$  randomly  
32 generated indices. Then the decoding algorithm is performed on  $\mathbf{x}^i$  using the set  
33  $\mathbf{J}$  to make it feasible. This means that, using the sequence  $\mathbf{J}$ , and one item/bit at  
34 a time all constraints are checked for capacity satisfaction, using the corresponding



1 column of the coefficient matrix of the resources. If all constraints are satisfied, the  
 2 bit 1 is maintained and the item is stored in the knapsack. Otherwise, the bit is  
 changed to 0. See Algorithm 2. Another decoding algorithm which starts from the

---

**Algorithm 2** Decoding algorithm used in IbAFSA

---

**Require:** Point  $\mathbf{x}^i$  and the set  $\mathbf{J} = \{J_1, J_2, \dots, J_n\}$

```

1: Set  $\text{sum}_k := 0$ , for  $k = 1, 2, \dots, m$ 
2: for  $j = 1$  to  $n$  do
3:   if  $x_{J_j}^i = 1$  then
4:     Set  $\text{flag} := 1$ 
5:     for  $k = 1$  to  $m$  do
6:       if  $\text{sum}_k + a_{k,J_j} > b_k$  then
7:         Set  $\text{flag} := 0$ 
8:         break
9:       end if
10:    end for
11:    if  $\text{flag} = 1$  then
12:      for  $k = 1$  to  $m$  do
13:        Set  $\text{sum}_k := \text{sum}_k + a_{k,J_j}$ 
14:      end for
15:    else
16:      Set  $x_{J_j}^i := 0$ 
17:    end if
18:  end if
19: end for
20: return Feasible point  $\mathbf{x}^i$ 

```

---

3  
 4 first index and sequentially continues can be applied but the obtained solution may  
 5 not be optimal.

6 After the decoding algorithm, a simple greedy-like heuristic called `add_item` (Al-  
 7 gorithm 3) is implemented to each feasible solution aiming to improve that point  
 8 without violating any constraint. When solving the single knapsack problem, this  
 9 heuristic operation uses the information of the *pseudo-utility* ratios,  $\delta_j$ , which are  
 10 defined as the ratios of the objective function coefficients ( $c_j$ 's) to the coefficients of  
 11 the single constraint ( $a_j$ 's). The greater the ratio, the higher the chance that the  
 12 corresponding variable will be equal to one in the solution [18]. In the generalization  
 13 of this `add_item` heuristic for the 0-1 MKP, the *pseudo-utility* ratios of every item  
 14 in every constraint are calculated, and only the lowest value for each item is con-  
 15 sidered (i.e.,  $\delta_j = \min\{(c_j b_k)/a_{k,j}\} \ j = 1, \dots, n, k = 1, \dots, m$ ). Then  $\delta_j$  are sorted  
 16 in decreasing order and a set  $\mathbf{J} = \{J_1, J_2, \dots, J_n\}$  is defined with the indices of the  
 17  $\delta_j$  in decreasing order. One item is added each time in the knapsack if it satisfies  
 18 all the constraints following the sequence of indices in the set  $\mathbf{J}$ . This procedure is

continued until the entire sequence of indices has been used.

---

**Algorithm 3** Add\_item algorithm used in IBAFSA

---

**Require:** Feasible point  $\mathbf{x}^i$  and set  $\mathbf{J} = \{J_1, J_2, \dots, J_n\}$

```

1: Compute  $\text{sum}_k = \sum_{j=1}^n a_{k,j}x_j^i$ , for  $k = 1, 2, \dots, m$ 
2: for  $j = 1$  to  $n$  do
3:   if  $x_{J_j}^i = 0$  then
4:     Set  $\text{flag} := 1$ 
5:     for  $k = 1$  to  $m$  do
6:       if  $\text{sum}_k + a_{k,J_j} > b_k$  then
7:         Set  $\text{flag} := 0$ 
8:         break
9:       end if
10:    end for
11:    if  $\text{flag} = 1$  then
12:      Set  $x_{J_j}^i := 1$ 
13:      for  $k = 1$  to  $m$  do
14:        Set  $\text{sum}_k := \text{sum}_k + a_{k,J_j}$ 
15:      end for
16:    end if
17:  end if
18: end for
19: return Improved feasible point  $\mathbf{x}^i$ 

```

---

1

2 *3.4. Selection of a new population*

3 Among the trial points  $\mathbf{y}^{i,t}$  and the current points  $\mathbf{x}^{i,t}$ ,  $i = 1, 2, \dots, N$ , at iteration  
4  $t$ , in order to decide whether or not they should become members of the population  
5 in the next iteration,  $t + 1$ , the trial point is compared to the current point using the  
6 following greedy criterion:

$$\mathbf{x}^{i,t+1} = \begin{cases} \mathbf{y}^{i,t} & \text{if } z(\mathbf{y}^{i,t}) \geq z(\mathbf{x}^{i,t}) \\ \mathbf{x}^{i,t} & \text{otherwise} \end{cases}, \quad i = 1, 2, \dots, N. \quad (4)$$

7 *3.5. Leaping behavior*

8 When the best objective function value in the population does not change for a  
9 certain number of iterations, the algorithm may have stagnated. The other points  
10 of the population will eventually converge to that objective function value. To be  
11 able to escape from this region and to try to converge to the optimal solution, the  
12 algorithm performs the leaping behavior, at every  $L$  iterations. In the leaping, a  
13 point  $\mathbf{x}^{\text{rand}}$  ( $\text{rand} \in \{1, 2, \dots, N\}$ ) is randomly selected from the current population  
14 and some randomly selected bits of the point are changed from 0 to 1 or vice versa

1 with probability  $p_m$ . The value  $p_m = 0.01$  is widely used in binary represented  
 2 methods. The described operation is similar to a mutation with probability  $p_m$  of  
 3 genetic/evolutionary algorithms.

4 Afterwards, decoding and the add\_item heuristic are implemented, and the new  
 5 point replaces the point  $\mathbf{x}^{\text{rand}}$ .

### 6 *3.6. Termination conditions*

7 Let  $T_{\text{max}}$  be the maximum number of iterations. Let  $z_{\text{max}}$  be the maximum  
 8 objective function value attained at iteration  $t$  and  $z_{\text{opt}}$  be the known optimal value  
 9 available in the literature. The proposed IbAFSA terminates if one of the conditions

$$t > T_{\text{max}} \text{ or } |z_{\text{max}} - z_{\text{opt}}| \leq \epsilon \quad (5)$$

10 holds, where  $\epsilon$  is a small positive tolerance. This condition enables the algorithm  
 11 to terminate when the best known solution with a tolerance  $\epsilon$  is reached; otherwise,  
 12 it continues execution until  $T_{\text{max}}$  is exceeded. However, if the optimal value of the  
 13 given problem is unknown, the algorithm may use other termination conditions.

### 14 *3.7. Reinitialization of the population*

15 Past experiments with bAFSA [43] have shown that, from a certain iteration on,  
 16 all the individual points in a population converge to a non-optimal point, even after  
 17 the leaping behavior has been performed. To diversify the search, we propose to  
 18 reinitialize the population randomly, every  $R$  iterations, keeping the best solution  
 19 found so far. In practical terms, this technique has greatly improved the quality of  
 20 the solutions and increased the consistency of the proposed improved binary version  
 21 of AFSA.

### 22 *3.8. The algorithm*

23 The pseudocode of the herein proposed improved binary version of AFSA for  
 24 solving the 0–1 multidimensional knapsack problem (1) is shown in Algorithm 4.

### 25 *3.9. Time complexity of one iteration of IbAFSA*

26 The algorithm time complexity is usually measured using  $\mathcal{O}$  notation and shows  
 27 how the amount of time needed to complete the (operations in the) algorithm varies  
 28 as the size of the input data  $m$  and  $n$  increase. The time complexity of one iteration,  
 29 for the worst-case scenario of the Algorithm 4, is analyzed assuming that we have a  
 30 population of  $N$  points, each point is represented by an  $n$ -vector and the problem  
 31 has  $m$  constraints. The computation for each iteration is as follows.

---

**Algorithm 4** IbaFSA

---

**Require:**  $T_{\max}$  and  $z_{\text{opt}}$  and other values of parameters

```
1: Set  $t := 1$ . Initialize population  $\mathbf{x}^{i,1}, i = 1, 2, \dots, N$ 
2: Perform decoding and add_item, evaluate the population and identify  $\mathbf{x}^{\max}$  and  $z_{\max}$ 
3: while ‘termination conditions are not met’ do
4:   if MOD(t,R)=0 then
5:     Reinitialize population  $\mathbf{x}^{i,t}, i = 1, 2, \dots, N - 1$ 
6:     Perform decoding and add_item, evaluate population and identify  $\mathbf{x}^{\max}$  and  $z_{\max}$ 
7:   end if
8:   for all  $\mathbf{x}^{i,t}$  do
9:     Compute ‘visual scope’ and ‘crowding factor’
10:    if ‘visual scope’ is empty then
11:      Perform random behavior to create trial point  $\mathbf{y}^{i,t}$ 
12:    else if ‘visual scope’ is not crowded then
13:      if  $z(\mathbf{x}^{\text{best}}) > z(\mathbf{x}^{i,t})$  then
14:        Perform chasing behavior to create trial point  $\mathbf{y}^{i,t}$ 
15:      else if  $z(\bar{\mathbf{x}}) > z(\mathbf{x}^{i,t})$  then
16:        Perform swarming behavior to create trial point  $\mathbf{y}^{i,t}$ 
17:      else if  $z(\mathbf{x}^{\text{rand}}) > z(\mathbf{x}^{i,t})$  then
18:        Perform searching behavior to create trial point  $\mathbf{y}^{i,t}$ 
19:      else
20:        Perform random behavior to create trial point  $\mathbf{y}^{i,t}$ 
21:      end if
22:    else if ‘visual scope’ is crowded then
23:      if  $z(\mathbf{x}^{\text{rand}}) > z(\mathbf{x}^{i,t})$  then
24:        Perform searching behavior to create trial point  $\mathbf{y}^{i,t}$ 
25:      else
26:        Perform random behavior to create trial point  $\mathbf{y}^{i,t}$ 
27:      end if
28:    end if
29:  end for
30:  Perform decoding and add_item to get  $\mathbf{y}^{i,t}, i = 1, 2, \dots, N$  and evaluate them
31:  Select new population  $\mathbf{x}^{i,t+1}, i = 1, 2, \dots, N$ 
32:  if MOD(t,L)=0 then
33:    Perform leaping behavior, decoding, add_item and evaluate
34:  end if
35:  Identify  $\mathbf{x}^{\max}$  and  $z_{\max}$ 
36:  Set  $t := t + 1$ 
37: end while
38: return  $\mathbf{x}^{\max}$  and  $z_{\max}$ 
```

---

- 1 Step 1, the initialization, takes  $Nn$  operations;
- 2 Step 2, decoding and add\_item take  $Nmn$  and evaluating the population takes  $N$ ;
- 3 the total time is  $N(mn + 1)$ ;

1 Step 4 – Step 7 take  $(N - 1)n$  (for reinitialization of  $N - 1$  points),  $(N - 1)mn$  for  
 2 decoding and `add_item`, and  $N - 1$  for evaluation, i.e., the total is  $(N - 1)(n +$   
 3  $mn + 1)$ ;  
 4 Step 8 – Step 29: to compute the ‘visual scope’ of each point and to check which  
 5 points are in its vicinity take  $n^2$ ; to generate the trial point takes  $n$ ; thus, when  
 6 all  $N$  points are considered, the total time is  $Nn^2 + Nn$ ;  
 7 Step 30 takes  $Nmn$ ;  
 8 Step 31 takes  $N$ ;  
 9 Step 32 – Step 34 take  $mn$ ;  
 10 Adding everything up  $Nn + N(mn + 1) + (N - 1)(n + mn + 1) + Nn(n + 1) + Nmn +$   
 11  $N + mn$  gives a time of  $N(3mn + 3n + 3 + n^2)$ . Considering that  $N$  is a constant, the  
 12 complexity is of  $\mathcal{O}(n^2)$  for fixed  $m$ ,  $\mathcal{O}(m)$  for fixed  $n$  and  $\mathcal{O}(mn + n^2)$  for variable  $m$   
 13 and  $n$ .

#### 14 4. Experimental Results

15 We code IbaFSA in C and compile with Microsoft Visual Studio 10.0 compiler in  
 16 a PC having 2.5 GHz Intel Core 2 Duo processor and 4 GB RAM. We set  $N = 100$ ,  
 17  $\delta = 0.5$ ,  $\theta = 0.8$ ,  $p_m = 0.01$  and  $\epsilon = 10^{-4}$ . In order to perform the leaping behavior,  
 18 we set  $L = \max(25, n)$ . After several experiments, we set the parameter  $R$  for the  
 19 reinitialization of the population to 100. We consider six benchmark sets of 0–1  
 20 MKP with a total of 55 instances from OR-library<sup>1</sup>. These problems are widely used  
 21 for the measurement of effectiveness of an algorithm in the optimization community.  
 22 The number of variables,  $n$ , in the instances varies from six to 105, and  $m$  (number of  
 23 constraints) varies from two to 30. Table 1 lists the values of  $n$  and  $m$  of the instances  
 24 for each problem set. Since they are benchmark instances, the optimal solution,  $z_{\text{opt}}$ ,  
 25 is known and the termination condition described in (5) can be used to terminate  
 26 the algorithm. For these instances, we set  $T_{\text{max}} = 1000$  if  $n \leq 50$ ; otherwise 2000.

27 First, we compare IbaFSA with CPLEX MIP solver, GA [18], bAFSA [43] and  
 28 GADS [19]. We run CPLEX MIP solver in our computer to solve the instances and  
 29 report the obtained results. We use the data of GA available in the corresponding  
 30 literature [18]. We note that GA uses a different termination condition and performs  
 31 just a single run for each instance. We also code GADS in C and run with the rec-  
 32 ommended parameters [19]. In GADS, partially matched crossover, bit flip mutation  
 33 and inversion are used. The crossover, mutation and inversion probabilities are set  
 34 to 0.9, 0.1 and 0.03 respectively. We use the same termination conditions (5) for

---

<sup>1</sup><http://people.brunel.ac.uk/~mastjjb/jeb/info.html>

Table 1: Values of  $(n, m)$  of each instance in the sets

Problem set	Number of instances	$(n, m)$
HP	2	(28, 4), (35, 4)
PB	6	(27, 4), (34, 4), (29, 2), (20, 10), (40, 30), (37, 30)
PT	7	(6, 10), (10, 10), (15, 10), (20, 10), (28, 10), (39, 5), (50, 5)
SENTO	2	(60, 30), (60, 30)
WEING	8	(28, 2), (28, 2), (28, 2), (28, 2), (28, 2), (28, 2), (105, 2), (105, 2)
WEISH	30	(30, 5), (30, 5), (30, 5), (30, 5), (30, 5), (40, 5), (40, 5), (40, 5), (40, 5), (50, 5), (50, 5), (50, 5), (50, 5), (60, 5), (60, 5), (60, 5), (60, 5), (60, 5), (70, 5), (70, 5), (70, 5), (70, 5), (80, 5), (80, 5), (80, 5), (80, 5), (80, 5), (90, 5), (90, 5), (90, 5), (90, 5), (90, 5), (90, 5)

1 GADS and run the program 30 times to report the results. We also run bAFSA and  
2 IbAFSA 30 times for each instance. In this comparison, GA, bAFSA, GADS and  
3 IbAFSA had the same value of  $N$ . The comparative results are shown in Table 2.

4 The table reports the average number of iterations, ‘AIT’, and the average com-  
5 putational time (in seconds), ‘AT’, considering all the 30 runs and only the successful  
6 runs, ‘Succ. runs’. If the algorithm finds the optimal solution (or near optimal ac-  
7 cording to an error tolerance) to an instance in a run, then the run is considered  
8 to be a successful one. Further, ‘ABT’ is the average best time to find the optimal  
9 value, i.e., is the average of the best time, from the 30 runs, among all instances of a  
10 set. ‘Nopt’ is the number of instances that were solved to optimality (at least in one  
11 run among the 30 runs) in a set, and ‘ASR’ is the average, among all instances in  
12 the set, of the success rate (in %). The success rate, ‘SR’, indicates the percentage  
13 of the 30 runs that found the known optimal solution according to the given error  
14 tolerance.

15 We note that CPLEX MIP and GA (in Table 2) solve all the instances to opti-  
16 mality in each set.

17 When comparing bAFSA with IbAFSA, we may conclude that IbAFSA gives  
18 better results than bAFSA, as far as ‘AIT’, ‘AT’ and ‘ASR’ for the 30 runs are  
19 concerned. See Table 2. The overall average success rate of IbAFSA is 92.97%,  
20 whereas that of bAFSA is 68.12%. This is obtained mainly due to the usage of  
21 add.item and reinitialization techniques in IbAFSA. Although ‘AIT’ and ‘AT’, for  
22 the ‘Succ. runs’, are in general smaller in bAFSA than in IbAFSA, those values  
23 correspond to a few number of successful runs (see column ‘ASR’). We note that

Table 2: Results obtained by CPLEX MIP, GA, bAFSA, GADS and IbaFSA

Prob. set	CPLEX MIP		GA		bAFSA						GADS				IbaFSA									
	1 run		1 run		30 runs			Succ. runs			30 runs				Succ. runs		30 runs			Succ. runs				
	AIT	AT	AT <sup>†</sup>	ABT <sup>†</sup>	AIT	AT	Nopt	ASR	AIT	AT	ABT	AIT	AT	Nopt	ASR	AIT	AT	AIT	AT	Nopt	ASR	AIT	AT	ABT
HP	32	0.05	2.6	0.4	737	2.10	2	28.33	74	0.26	0.09	399	0.22	2	76.67	235	0.13	189	0.40	2	98.33	176	0.37	0.06
PB	526	0.13	5.2	0.1	452	1.35	6	60.56	96	0.31	0.08	352	0.25	6	78.33	183	0.13	77	0.17	6	100.00	77	0.17	0.02
PT	22	0.06	3.2	0.2	324	1.19	7	69.52	56	0.25	0.14	335	0.24	5	71.43	70	0.04	262	0.83	7	76.19	123	0.39	0.18
SENTO	953	0.12	11.5	0.3	1712	9.95	2	15.00	84	0.61	0.50	1959	3.03	1	6.67	1379	1.92	43	0.28	2	100.00	43	0.28	0.13
WEING	12	0.14	4.3	0.4	602	4.22	8	65.83	80	0.48	0.30	665	0.76	6	70.33	184	0.09	543	3.11	8	78.75	266	1.30	0.51
WEISH	21	0.07	6.4	0.1	468	2.84	30	76.11	67	0.47	0.34	1312	1.38	17	33.33	493	0.41	109	0.56	30	98.44	89	0.45	0.08
Average	109	0.09	5.6	0.2	522	2.90		68.12	71	0.42	0.28	979	1.04		49.15	345	0.29	188	0.91		92.97	119	0.53	0.16

<sup>†</sup> Not applicable here (due to different machine used), only to show

1 IbaFSA performs favorably relative to the criterion ‘ABT’ when compared with  
 2 bAFSA. Based on ‘Nopt’, we may conclude that bAFSA and IbaFSA also solve all  
 3 instances to optimality (at least in one run among the 30). Using reinitialization of  
 4 the population, IbaFSA gives more successful runs after  $R$  iterations.

5 When we compare the results of IbaFSA with those of GADS in Table 2, we  
 6 observe that IbaFSA has a better performance than GADS relative to the criterion  
 7 ‘AIT’, in almost all sets. The average computational time (‘AT’) of IbaFSA is higher  
 8 than that of GADS in some sets. This is due to the procedure that aims to identify  
 9 the points inside the ‘visual scope’ of each individual point, at all iterations. We note  
 10 that GADS did not reach the optimal solution to some instances in any of the 30  
 11 runs. The ‘ASR’ obtained by GADS for each set are all under 80%, while IbaFSA  
 12 reaches 100% in sets PB and SENTO, and almost 99% in sets HP and WEISH.

13 Since GADS and IbaFSA are population-based stochastic methods, we compare  
 14 them further using different performance criteria: the average of the obtained objec-  
 15 tive function values, ‘ $z_{avg}$ ’, with the standard deviation of the function values, ‘ $std$ ’,  
 16 and the success rate, ‘SR’ (in %). For a clear comparison, we plot in Figs. 1–5 the  
 17 bar profiles of ‘ $z_{avg}$ ’ obtained by GADS and IbaFSA among the 30 runs, for all the  
 18 instances of the six sets. The values of ‘ $std$ ’ relative to ‘ $z_{avg}$ ’ are shown over the  
 19 bars. We can notice that IbaFSA outperforms GADS. The ‘ $z_{avg}$ ’ values of GADS  
 20 are in general slightly smaller than those of IbaFSA, in particular on instances 16  
 to 30 of the WEISH set, and the ‘ $std$ ’ values are larger with GADS. To emphasize

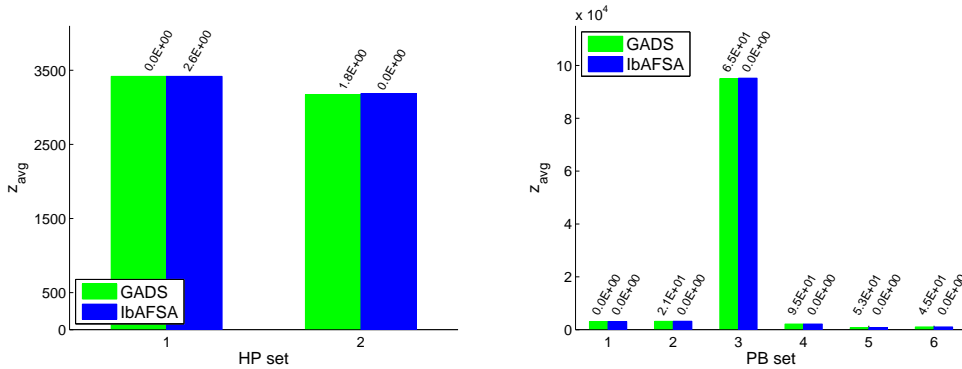


Figure 1: Comparison of  $z_{avg}$  and  $std$  on HP set (left) and PB set (right)

21 the improvement on the quality of the solutions obtained by the herein proposed  
 22 IbaFSA, when compared with the preliminary version bAFSA [43] and GADS, we  
 23 show in Fig. 6 the bar profiles corresponding to ‘SR’. We may conclude that IbaFSA  
 24 outperforms GADS and bAFSA in criterion ‘SR’.  
 25



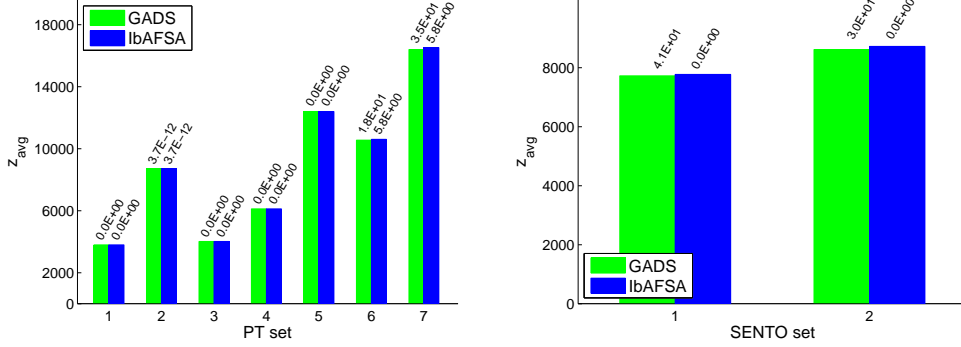


Figure 2: Comparison of  $z_{avg}$  and  $std$  on PT set (left) and SENTO set (right)

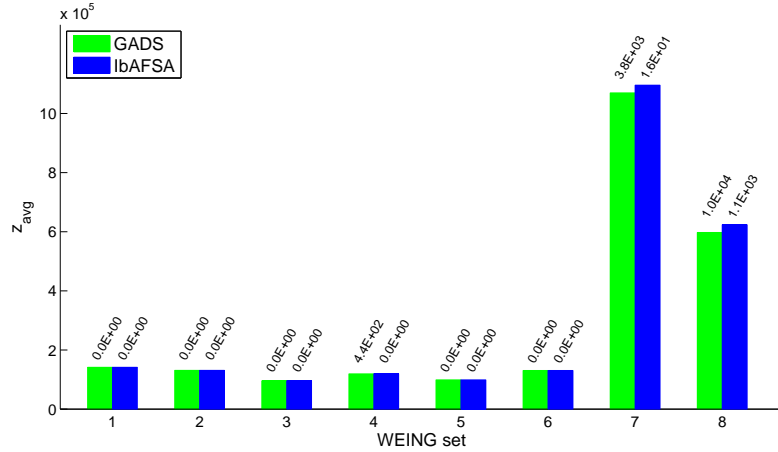


Figure 3: Comparison of  $z_{avg}$  and  $std$  on WEING set

1 In the previous section, the time complexity of one iteration of IbAFSA has been  
 2 analyzed. Although the running time, per iteration, of IbAFSA could be greater  
 3 than that of GADS, which is of order of complexity  $\mathcal{O}(mn)$ , the number of iterations  
 4 required to reach the optimal (or near optimal) solution is much smaller with IbAFSA  
 5 – an average over the 55 instances of 119 in IbAFSA against 345 in GADS – and it  
 6 reaches the optimal solution more often – an average of 188 iterations with 92.97% of  
 7 successful cases in IbAFSA against 979 iterations with 49.15% of successes in GADS,  
 8 over the 30 runs (see Table 2).

9 We now compare IbAFSA with HGA (Hybrid Genetic Algorithm) described  
 10 in [20]. For a fair comparison we run IbAFSA with  $N = 5n$  and  $T_{max} = 3000$

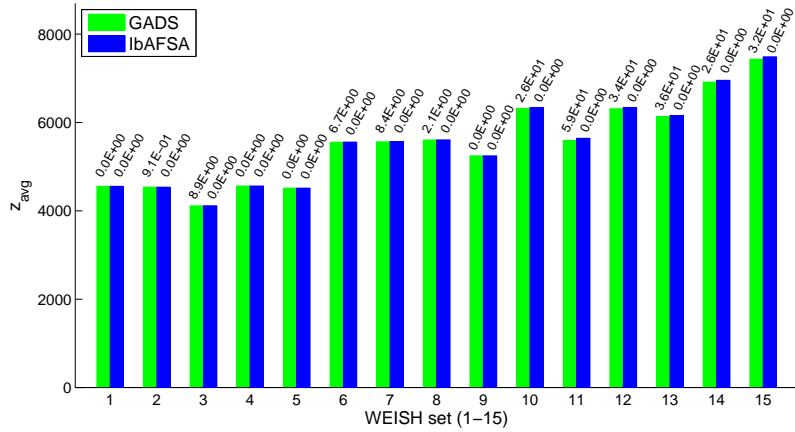


Figure 4: Comparison of  $z_{avg}$  and  $std$  on instances 1 to 15 from WEISH set

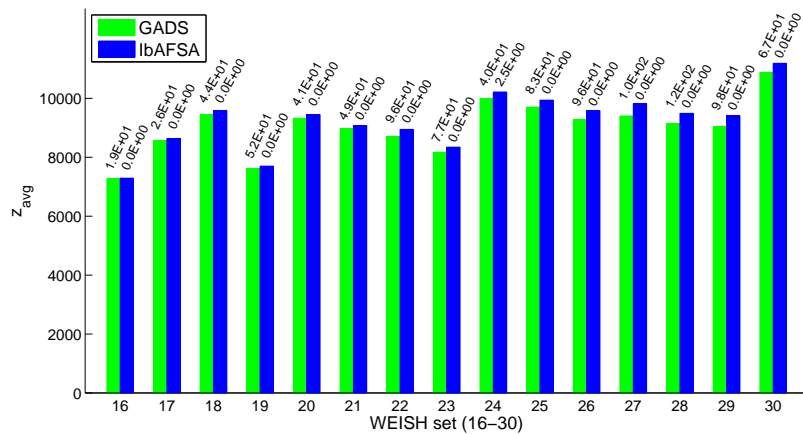


Figure 5: Comparison of  $z_{avg}$  and  $std$  on instances 16 to 30 from WEISH set

1 for all the 55 instances and 10 independent runs were carried out for each instance  
 2 (same as HGA). Table 3 shows the comparison based on the different performance  
 3 criteria. The data of HGA are taken from the corresponding literature. Although  
 4 the machines used to obtain the results are different, IbaFSA shows a very good  
 5 performance when compared with HGA.

6 We also compare IbaFSA with some variants of the particle swarm optimization  
 7 (PSO) algorithm. Table 4 contains numerical results obtained by binary versions of  
 8 PSO from the literature. The results are taken from [38, 40]. SBPSO is a Set-Based

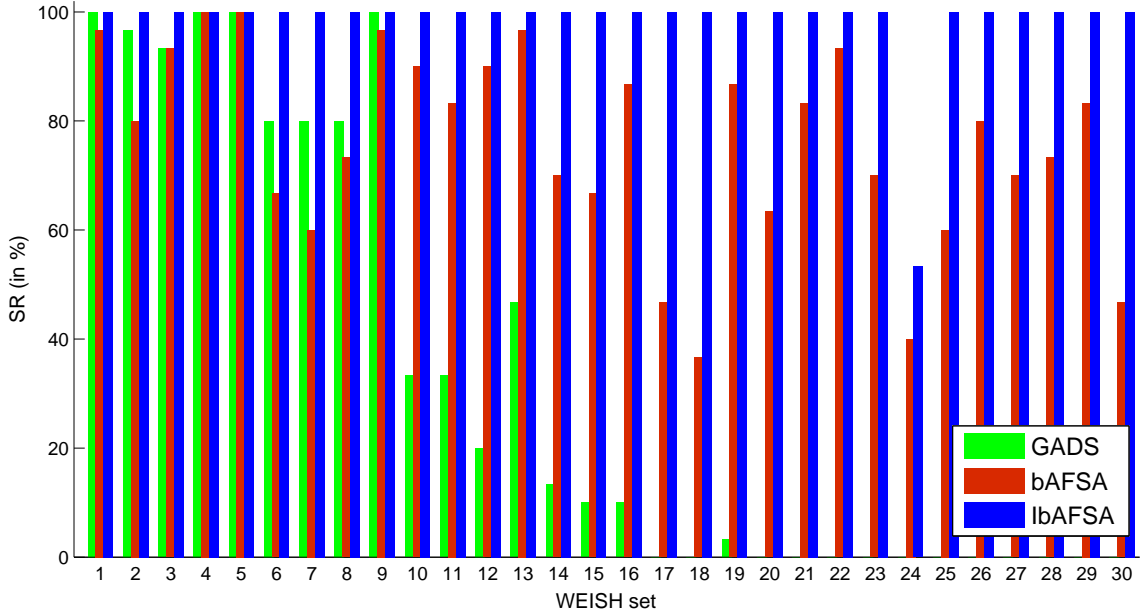


Figure 6: Comparison of SR: IbAFSA *vs.* GADS and bAFSA

1 PSO proposed in [40] that handles the constraints using a dead penalty. On the  
 2 other hand, the modified binary PSO (MBPSO) in [38] applies a static penalty func-  
 3 tion approach to handle the constraints and uses a probability function to maintain  
 4 diversity in the swarm. The results of BPSO are taken from the last cited paper.  
 5 The comparisons are based on the average success rate, ‘ASR’, and average percent-  
 6 age gap to optimality, ‘APG’, where the percentage gap to the optimality (‘PG’) is  
 7 defined by

$$PG = \frac{z_{\text{opt}} - z_{\text{max}}}{z_{\text{opt}}} 100\%.$$

8 The table shows that IbAFSA outperforms the three selected PSO versions in both  
 9 criteria.

10 Finally, we compare the proposed IbAFSA with the heuristic methods HDP+LBC  
 11 (Hybrid Dynamic Programming with Limited Branch-and-Cut) and DPHEU (Dom-  
 12 inance Principle based Heuristic) described in [24, 27]. The results for comparison  
 13 are reported in Table 5. This comparison is based on ‘PG’ and on ‘Nopt’. We remark  
 14 that the reported values for HDP+LBC and DPHEU are relative to just one run, and

Table 3: Comparison of HGA and IbAFSA

Prob. set	HGA			IbAFSA		
	ABT	AT	Nopt	ABT	AT	Nopt
HP	61.0	84.0	2	0.19	0.82	2
PB	108.0	425.0	6	0.08	0.24	6
PT	150.0	352.0	7	0.51	1.49	7
SENTO	29.0	162.0	2	0.55	0.90	2
WEING	126.0	452.0	8	4.46	26.04	8
WEISH	101.0	321.0	30	0.95	2.69	30
Average	95.8	299.3		1.27	5.53	

Table 4: Comparison of BPSO, SBPSO, MBPSO and IbAFSA

Prob. set	ASR				APG			
	SBPSO	BPSO	MBPSO	IbAFSA	SBPSO	BPSO	MBPSO	IbAFSA
HP	5.00	25.00	53.00	98.33	1.68	0.97	0.33	0.01
PB	14.50	21.50	48.17	100.00	1.81	1.35	0.56	0.00
PT	59.00	–	–	76.19	0.32	–	–	0.03
SENTO	13.50	27.00	48.00	100.00	0.45	0.22	0.10	0.00
WEING	47.50	57.50	78.13	78.75	2.49	0.34	0.11	0.02
WEISH	32.23	56.30	74.20	98.44	0.66	0.80	0.53	0.00
Average	34.25	49.63	69.63	92.97	1.04	0.78	0.44	0.01

– Not considered

1 the reported IbAFSA values correspond to the percentage gap to optimality of the  
2 best run (‘PG best’) and the average percentage gap, over the 30 runs, ‘APG’. We  
3 note that these values are averaged over all the instances in a set. IbAFSA solves all  
4 the instances in the sets HP, PB, PT, SENTO, WEING and WEISH to optimality  
5 in at least one of the 30 runs. From the table, we conclude that IbAFSA gives good  
6 performance when compared with HDP+LBC and DPHEU.

7 Although other solution methods are available, they are not considered during  
8 the numerical comparisons with IbAFSA since they use just a few instances of each  
9 set. Based on the numerical experiments carried out until now, we may conclude  
10 that the proposed improved binary version of the artificial fish swarm algorithm is  
11 rather effective and competitive when solving the 0–1 MKP.

Table 5: Comparison of HDP+LBC, DPHEU and IbaAFSA

Prob. set	HDP+LBC		DPHEU		IbaAFSA		
	PG	Nopt	PG	Nopt	PG best	APG	Nopt
HP	0.45	–	0.00	2	0.00	0.01	2
PB	0.20	–	0.04	5	0.00	0.00	6
PT	0.02	–	0.00	7	0.00	0.03	7
SENTO	0.00	–	0.00	2	0.00	0.00	2
WEING	0.00	–	0.00	8	0.00	0.02	8
WEISH	–	–	0.03	28	0.00	0.00	30
Average	0.09		0.02		0.00	0.01	

– Not considered/not available in the paper

## 1 5. Conclusions

2 In this paper, an improved binary version of the artificial fish swarm algorithm  
3 for solving 0–1 MKP has been presented. In this method a point in the population  
4 is represented by a binary string of 0/1 bits. The Hamming distance is computed in  
5 order to identify the neighborhood points inside the ‘visual scope’ of a current point.  
6 Depending on the number of points inside the ‘visual scope’, the current point can  
7 perform either chasing, swarming, searching or random behavior. To create a trial  
8 point, each bit is generated by copying the corresponding bit from the current point  
9 or from some other specified point, with equal probability. When a leaping behavior  
10 is to be performed, some randomly chosen bits of a selected point are changed from 0  
11 to 1, or 1 to 0, with an user defined probability. The decoding algorithm, combined  
12 also with an add.item algorithm to each feasible point, are also implemented in  
13 order to make the points feasible and improve the quality of the solution. A greedy  
14 selection criterion is used to decide whether or not the trial points should become  
15 members of the population in the next iteration. A periodic reinitialization of the  
16 population has shown to improve the quality of the solutions and increase IbaAFSA  
17 consistency.

18 A comparison of IbaAFSA with other solution methods available in the litera-  
19 ture has been carried out with the 55 benchmark test instances. The effective-  
20 ness of IbaAFSA has been shown when compared with a previous binary version of  
21 AFSA. The occurrence of obtaining the optimal solution has increased in average  
22 from 68.12% up to 92.97%. The running time of one iteration of IbaAFSA for solving  
23 0–1 MKP with  $m$  constraints and  $n$  decision variables has been analyzed and the  
24 complexity is of  $\mathcal{O}(n^2)$  for fixed  $m$ ,  $\mathcal{O}(m)$  for fixed  $n$  and  $\mathcal{O}(mn + n^2)$  for variable  $m$   
25 and  $n$ . The comparison with GADS and HGA became highly favorable to IbaAFSA.

1 The results show that IbaFSA outperforms GADS in all tested criteria ('AIT', 'AT',  
2 'Nopt' and 'ASR') and HGA in the criteria 'ABT' and 'AT'. The numerical results  
3 demonstrate the superiority of the proposed IbaFSA against swarm-based heuristics  
4 such as some binary versions of the PSO algorithm, BPSO, MBPSO and SBPSO.  
5 From the comparison with the heuristics HDP+LBC and DPHEU, we conclude that  
6 IbaFSA has a competitive performance. Thus, it is found that the proposed method  
7 is rather effective and competitive when solving small benchmark instances. Future  
8 developments will focus on the large instances of the 0–1 MKP and on the simplifi-  
9 cation of some procedures, in particular those related with generating a trial point  
10 from the current one based on fish behavior, in order to reduce the processing time.

## 11 Acknowledgments

12 The authors would like to thank four anonymous referees for their careful reading of the paper  
13 and for their helpful comments and suggestions which greatly improved the paper. The first author  
14 acknowledges Ciência 2007 of FCT (Foundation for Science and Technology), Portugal for the  
15 fellowship grant: C2007-UMINHO-ALGORITMI-04. Financial support from FEDER COMPETE  
16 (Operational Programme Thematic Factors of Competitiveness) and FCT under project FCOMP-  
17 01-0124-FEDER-022674 is also acknowledged.

## 18 References

- 19 [1] Petersen CC. Computational experience with variants of the balas algorithm  
20 applied to the selection of R&D projects. *Management Science* 13(9) (1967)  
21 736–750.
- 22 [2] Weingartner HM. *Mathematical Programming and the Analysis of Capital Bud-*  
23 *geting Problems* Prentice-Hall, Englewoods Cliffs, 1963.
- 24 [3] Gavish B, Pirkul H. Efficient algorithms for solving multiconstraint zero–one  
25 knapsack problems to optimality. *Mathematical Programming* 31 (1985) 78–  
26 105.
- 27 [4] Shih W. A branch and bound method for the multiconstraint zero–one knapsack  
28 problem. *Journal of the Operational Research Society* 30 (1979) 369–378.
- 29 [5] Pisinger D. Algorithms for knapsack problems. Ph.D. thesis, Department of  
30 Computer Science, University of Copenhagen, Denmark, 1995.  
31 <http://www.diku.dk/hjemmesider/ansatte/pisinger/>

- 1 [6] Balev S, Yanev N, Fréville A, Andonov R. A dynamic programming based  
2 reduction procedure for the multidimensional 0–1 knapsack problem. *European*  
3 *Journal of Operational Research* 166 (2008) 63–76.
- 4 [7] Weingartner HM, Ness DN. Methods for the solution of the multidimensional  
5 0/1 knapsack problem. *Operations Research* 15 (1967) 83–103.
- 6 [8] Fréville A, Plateau G. The 0–1 bidimensional knapsack problem: Towards an  
7 efficient high-level primitive tool. *Journal of Heuristics* 2 (1996) 147–167.
- 8 [9] Cabot AV. An enumeration algorithm for knapsack problems. *Operations Re-*  
9 *search* 18 (1970) 306–311.
- 10 [10] Schilling KE. The growth of  $m$ -constraint random knapsacks. *European Journal*  
11 *of Operational Research* 46 (1990) 109–112.
- 12 [11] Fontanari JF. A statistical analysis of the knapsack problem. *Journal of Physics*  
13 *A: Mathematical and General* 28 (1995) 4751–4759.
- 14 [12] Soyster AL, Lev B, Slivka W. Zero–one programming with many variables and  
15 few constraints. *European Journal of Operational Research* 2 (1978) 195–201.
- 16 [13] Lin FT. On the generalized fuzzy multiconstraint 0–1 knapsack problem. in:  
17 *Proceedings of the 2006 IEEE International Conference on Fuzzy Systems*. pp.  
18 984–989, 2006.
- 19 [14] Puchinger J, Raidl GR, Pferschy U. The multidimensional knapsack problem:  
20 structure and algorithms. *INFORMS Journal of Computing* 22(2) (2010) 250–  
21 265.
- 22 [15] Drexel A. A simulated annealing approach to the multiconstraint zero–one knap-  
23 sack problem. *Computing* 40 (1988) 1–8.
- 24 [16] Hanafi S, Fréville A. An efficient tabu search approach for the 0–1 multidimen-  
25 sional knapsack problem. *European Journal of Operational Research* 106 (1998)  
26 659–675.
- 27 [17] Vasquez M, Vimont Y. Improved results on the 0–1 multidimensional knapsack  
28 problem. *European Journal of Operational Research* 165 (2005) 70–81.
- 29 [18] Chu PC, Beasley JE. A genetic algorithm for the multidimensional knapsack  
30 problem. *Journal of Heuristics* 4 (1998) 63–86.

- 1 [19] Sakawa M, Kato K. Genetic algorithms with double strings for 0–1 programming  
2 problems. *European Journal of Operational Research* 144 (2003) 581–597.
- 3 [20] Djannaty F, Doostdar S. A hybrid genetic algorithm for the multidimensional  
4 knapsack problem. *International Journal of Contemporary Mathematical Sci-*  
5 *ences* 3(9) (2008) 443–456.
- 6 [21] Kong M, Tian P, Kao Y. A new ant colony optimization algorithm for the mul-  
7 tidimensional knapsack problem. *Computers & Operations Research* 35 (2008)  
8 2672–2683.
- 9 [22] Zou D, Gao L, Li S, Wu Z. Solving 0–1 knapsack problem by a novel global  
10 harmony search algorithm. *Applied Soft Computing* 11 (2011) 1556–1564.
- 11 [23] Akçay Y, Li H, Xu SH. Greedy algorithm for the general multidimensional  
12 knapsack problem. *Annals of Operations Research* 150 (2007) 17–29.
- 13 [24] Boyer V, Elkihel M, Baz DE. Heuristics for the 0–1 multidimensional knapsack  
14 problem. *European Journal of Operational Research* 199 (2009) 658–664.
- 15 [25] Hill RR, Cho YK, Moore JT. Problem reduction heuristic for the 0–1 multi-  
16 dimensional knapsack problem. *Computers & Operations Research* 39 (2012)  
17 19–26.
- 18 [26] Pirkul H. A heuristic solution procedure for the multiconstraint zero-one knap-  
19 sack problem. *Naval Research Logistics* 34 (1987) 161–172.
- 20 [27] Veni KK, Balachandar SR. A new heuristic approach for large size zero–one  
21 multi knapsack problem using intercept matrix. *International Journal of Com-*  
22 *putational and Mathematical Sciences* 4(5) (2010) 259–263.
- 23 [28] Fréville A. The multidimensional 0–1 knapsack problem: An overview. *European*  
24 *Journal of Operational Research* 155 (2004) 1–21.
- 25 [29] Jiang M, Mastorakis N, Yuan D, Lagunas MA. Image segmentation with im-  
26 proved artificial fish swarm algorithm. in: Mastorakis N et al. (eds.), *ECC 2008,*  
27 *LNEE*, Vol. 28. Springer-Verlag, Heidelberg, pp. 133–138, 2009.
- 28 [30] Jiang M, Wang Y, Pfletschinger S, Lagunas MA, Yuan D. Optimal multiuser  
29 detection with artificial fish swarm algorithm. in: Huang DS et al. (eds.), *ICIC*  
30 *2007, CCIS*, Vol. 2. Springer-Verlag, Heidelberg, pp. 1084–1093, 2007.



- 1 [31] Wang CR, Zhou C-L, Ma J-W. An improved artificial fish swarm algorithm and  
2 its application in feed-forward neural networks. in: Proceedings of the Fourth  
3 International Conference on Machine Learning and Cybernetics. pp. 2890–2894,  
4 2005.
- 5 [32] Wang X, Gao N, Cai S, Huang M. An artificial fish swarm algorithm based and  
6 abc supported QoS unicast routing scheme in NGI. in: Min G et al. (eds.), ISPA  
7 2006, LNCS, Vol. 4331. Springer-Verlag, Heidelberg, pp. 205–214, 2006.
- 8 [33] Rocha AMAC, Fernandes EMGP, Martins TFMC. Novel fish swarm heuristics  
9 for bound constrained global optimization problems. in: Murgante B et  
10 al. (eds.), Computational Science and Its Applications, ICCSA 2011, Part III,  
11 LNCS, Vol. 6784. Springer-Verlag, Heidelberg, pp. 185–199, 2011.
- 12 [34] Rocha AMAC, Martins TFMC, Fernandes EMGP. An augmented Lagrangian  
13 fish swarm based method for global optimization. *Journal of Computational and*  
14 *Applied Mathematics* 235 (2011) 4611–4620.
- 15 [35] Neshat M, Sepidnam G, Sargolzaei M, Toosi AN. Artificial fish swarm algorithm:  
16 a survey of the state-of-the-art, hybridization, combinatorial and indicative ap-  
17 plications. *Artificial Intelligence Review* (2012). DOI:10.1007/s10462-012-9342-2
- 18 [36] Mirjalili S, Lewis A. S-shaped versus V-shaped transfer functions for binary  
19 particle swarm optimization. *Swarm & Evolutionary Computation* 9 (2013) 1–  
20 14.
- 21 [37] Suresh K, Kumarappan N. Hybrid improved binary particle swarm optimization  
22 approach for generation maintenance scheduling problem. *Swarm & Evolution-*  
23 *ary Computation* 9 (2013) 69–89.
- 24 [38] Bansal JC, Deep K. A modified binary particle swarm optimization for knapsack  
25 problems. *Applied Mathematics and Computation* 218(22) (2012) 11042–11061.
- 26 [39] Beheshti Z, Shamsuddin SM, Yuhaniz SS. Binary accelerated particle swarm  
27 algorithm (BAPSA) for discrete optimization problems. *Journal of Global Op-*  
28 *timization* (2013). DOI 10.1007/s10898-012-0006-1
- 29 [40] Langeveld J, Engelbrecht AP. A generic set-based particle swarm optimization  
30 algorithm. in: Proceedings of the 2011 International Conference on Swarm In-  
31 telligence, France 2011, pp. id-1–id-10.

- 1 [41] Langeveld J, Engelbrecht AP. Set-based particle swarm optimization applied to  
2 the multidimensional knapsack problem. *Swarm Intelligence* 6 (2012) 297–342.
- 3 [42] Wang L, Wang X, Fu J, Zhen L. A novel probability binary particle swarm  
4 optimization algorithm and its application. *Journal of Software* 3(9) (2008) 28–  
5 35.
- 6 [43] Azad MAK, Rocha AMAC, Fernandes EMGP. Solving multidimensional 0-1  
7 knapsack problem with an artificial fish swarm algorithm. in: Murgante, B. et  
8 al. (eds.), *Computational Science and Its Applications, ICCSA 2012, Part III*,  
9 LNCS, Vol. 7335. Springer-Verlag, Heidelberg, pp. 72–86, 2012.
- 10 [44] Moraglio A, Johnson CG. Geometric generalization of the Nelder-Mead algo-  
11 rithm. in: Cowling P, Merz P (eds.), *EvoCOP 2010, LNCS, Vol. 6022*. Springer-  
12 Verlag, Heidelberg, pp. 190–201, 2010.
- 13 [45] Goldberg DE. *Genetic Algorithms in Search, Optimization, and Machine Learn-*  
14 *ing*. Addison-Wesley, Reading, MA, 1989.
- 15 [46] Michalewicz Z. *Genetic Algorithms+Data Structures=Evolution Programs*.  
16 Springer, Berlin, 1996.